Implementation, Experiments and Improvement of Optimal Trust System Placement in Smart Grid SCADA Networks

Faryad Abolhassani

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
February 2019
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Hadi Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Alexander Chefranov
2. Assoc. Prof. Dr. Gürcü Öz
3. Assoc. Prof. Dr. Mehtap Kose Ulukok

ABSTRACT

The aim of this thesis is to investigate the optimal Trust System Placement (TSP) method for smart grid Supervisory Control And Data Acquisition (SCADA) networks. At present, as SCADA networks are connected to the internet the scope of cyber-security concerns becomes much wider. Trust Systems (TSs) are deployed to provide the cyber-security of SCADA networks. TS are used to detect and block malicious activities. Optimal TSP problem is used to minimize the cost and maximize the security by selecting minimum number of TSs. Segmentation is the main part of the optimal TSP problem. Segmentation is used to divide the SCADA graph to small segments and ideal segmentation problem uniforms the size of the segment. Linear Programming Problem (LPP) is used to assign the TSs to some nodes among the bordering nodes and its constraint is that all inter-segment links are connected to at least one trust node.

The experiments are conducted on five IEEE test system topologies. The IEEE test system is categorized into small and large networks. The obtained results show that by increasing the quantity of segments the required quantity of TSs increases and small networks are more balanced than the large networks in size of segments.

In optimal TSP problem TS number per segments are not uniform over the segments. It may deteriorate the security of the segments and more delays happened to the inter-segment links. We propose optimal TSP uniformity problem to maximize the security and minimize the operational expenditure by distributing TSs over the segments.

After executing the optimal TSP uniformity on IEEE BUS 14, the distribution of TSs over the segments is improved by 100% with 3 segments and 5 segments and it is improved by 81% with 4 segments and it is improved by 35% with 6 segments.

Keywords: Smart grid Supervisory Control and Data Acquisition (SCADA) network, cyber-security, trust node, Minimum Spanning Tree (MST), Linear Programming Problem (LPP), segment, bordering node, uniformity problem.

ÖZ

Bu tezin amacı, akıllı şebeke Denetleme Kontrolü ve Veri Toplama (DKVVT) ağları için optimal Güven Sistemi Yerleştirme (GSY) yöntemini araştırmaktır. Şu anda, DKVVT ağları internete bağlı olduğundan, siber güvenlik endişelerinin kapsamı daha da genişlemektedir. DKVVT ağlarının siber güvenliğini sağlamak için Güven Sistemleri (GS'ler) kullanılmaktadır. GS, kötü niyetli etkinlikleri tespit etmek ve engellemek için kullanılır. En düşük GS sayısını seçerek maliyeti en aza indirmek ve güvenliği en üst düzeye çıkarmak için en uygun GSY sorunu kullanılır. Segmentasyon, optimum GSY probleminin ana parçasıdır. Segmentasyon, DKVVT grafiğini küçük parçalara bölmek için kullanılır ve ideal segmentasyon sorunu, segmentin boyutunu düzenler. Doğrusal Programlama Sorunu (DPS), GS'leri sınırlayıcı düğümler arasındaki bazı düğümlere atamak için kullanılır ve sınırlandırması, tüm bölümler arası bağlantıların en az bir güven düğümüne bağlı olmasıdır.

Deneyler beş IEEE test sistemi topolojisi üzerinde gerçekleştirilmiştir. IEEE test sistemi küçük ve büyük ağlara ayrılmıştır. Elde edilen sonuçlar, segmentlerin miktarını artırarak, gerekli GS'lerin miktarının arttığını ve küçük ağların, segment boyutundaki büyük ağlardan daha dengeli olduğunu göstermektedir.

Optimum GSY probleminde segment başına GS sayısı segmentler üzerinde aynı değildir. Segmentlerin güvenliğini bozabilir ve bölümler arası bağlantılarda daha fazla gecikme yaşanabilir. Güvenliği en üst seviyeye çıkarmak ve GS'leri segmentlere dağıtarak işletme giderlerini en aza indirmek için optimal GSY tekdüzelik problemi öneriyoruz.

IEEE BUS 14'te optimum GSY homojenliği uygulandıktan sonra, GS'lerin segmentler üzerindeki dağılımı, 3 segment ve 5 segment 100%, 4 segment 81% ve 6 segment 35% arttırılmıştır.

Anahtar Kelimeler: Denetleme kontrolü ve veri toplama (DKVVT) ağı, siber güvenlik, güven düğümü, Minimum yayılma ağacı (MYA), doğrusal programlama sorunu (DPS), segment, sınırlayıcı düğüm, tekdüzelik sorunu.

ACKNOWLEDGMENT

I would like to record my gratitude to Assoc. Prof. Dr. Alexander Chefranov for his supervision, advice, and guidance from the very early stage of this thesis as well as giving me extraordinary experiences throughout the work. Above all and the most needed, he provided me constant encouragement and support in various ways.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACL	Access Control List
ASCII	American Standard Code for Information Interchange
IDS	Intrusion Detection System
KV	Kilo Volt
LPP	Linear Programming Problem
MCA	Monitor-Control Attack
MST	Minimum Spanning Tree
MVAR	Mega Volt Ampere Reactive
MW	Mega Watt
PC	Personal Computer
PMU	Phasor Measurement Unit
PDC	Phasor Data Concentrator
PKI	Public Key Infrastructure
RSD	Relative Standard Deviation
SCADA	Supervisory Control And Data Acquisition
TS	Trust System
TSP	Trust System Placement
WAKE	Wide Area Key Exchange
WAMS	Wide Area Measurement System
WAN	Wide Area Network

Chapter 1

INTRODUCTION

Supervisory Control and Data Acquisition (SCADA) systems are used to control and monitor infrastructure of industries such as transmission and distribution networks of electricity, refineries, nuclear power plants. At present, industrial control systems and information technology (IT) systems merge together. As a consequence, SCADA and its equipment are a vulnerable target of the intrusion [1].

To keep the integrity of information and avoid cyber intrusions, smart grids need to deploy trust systems to monitor and control the input and output traffic. Trust systems are a combination of firewall and/or intrusion detection systems [2].

In this thesis, we are going to address the optimal trust system placement problem by focusing on discussion of its concepts, algorithms, and experimental results mainly from [1], and implementation of the optimal trust system placement scheme [1], experiments on IEEE test system topologies used in [1], and improvement of optimal trust system placement in smart grid SCADA networks [1], [3] and [4].

Adding trust systems to all the nodes in the grid is expensive and increases the processing time of the communication of nodes. In order to make it practical, networks must be divided into small networks, known as segments [1], and the selected bordering nodes [1] should be equipped by the trust systems. These nodes are called trust nodes. Inter-segment links are connected to the TS, hence, trust nodes are able to monitor ingress and egress traffic. Segmentation problems are often solved using mixed integer linear programming [5]. Uniformity of trust nodes among the segments is used to keep the balancing of dispersion of the trust nodes among the segments. As a consequence, the trust system is transferred from the oversized segment to the undersized segment if there is a link between the trust node in the oversized segment and the node that is not equipped by the trust system in the undersized segment.

The rest of this thesis is organized as follows. Chapter 2 discusses the related work mainly based on [1] and defines the problems of the thesis. Chapter 3 is about the design and implementation of the trust system placement problem and testing of the optimal trust system placement in smart grid of SCADA networks. Chapter 4 is about the improvement of the optimal trust system placement program using uniformity of trust systems distribution over network segments. Chapter 5 presents our experiments on IEEE test system topologies in the conditions of [1]. Chapter 6 is the conclusion. Appendix A shows the source codes of the program. Appendix B shows the databases of the IEEE test system topologies used in the experiments. Appendix C shows the screenshots of the experimental results on IEEE test system topologies.

Chapter 2

RELATED WORK AND PROBLEM DEFINITION

This chapter explains the terms related to trust system placement in smart grid networks (Section 2.1) and describes a method of trust nodes placement from [1] (Section 2.2). Thesis problems are defined in detail (Section 2.3).

2.1 Cyber-Security Terms and Notions for Smart Grid SCADA Networks

If infrastructures of smart grid network have not appropriate security equipment, it may result in vulnerabilities in power infrastructures. Power systems have three different types of sectors: generation, distribution and control, and consumption that may be the targets of the cyber-intrusion. Depending on the target, there can be three different cyber-attacks [5] as shown in Figure 1.

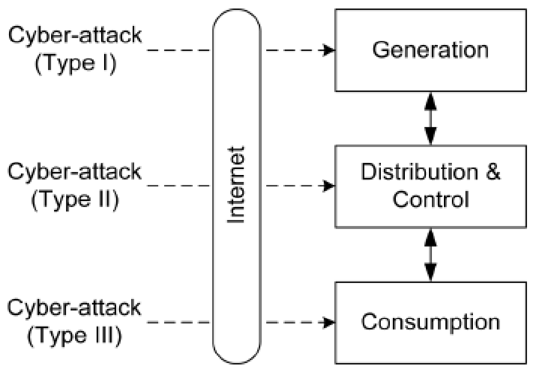


Figure 1: Three types of cyber-attacks on the electric grid through the internet [5]

Type I cyber-intrusion targets generation which intruder tries for interrupting or getting control of the operation of generators. Type II targets power distribution and control. In this type intruder makes failure in the supply of electricity or damaging the network equipment. Type III attacks target the consumption part. It causes an increment of load that damages the grids and it can bring down the grid. Password-secured access, group key ciphering for multicast transmissions, private key ciphering for unicast transmissions, user confirmation, message verification codes and firewalling of SCADA data traffic are the defense mechanisms to those attacks which are performed by trust systems.

Trust system validates input, identifies risks and bad data, and initiates appropriate alerts. It then assigns data types for the good data elements in each packet. It next determines if the recipient is authorized to read all of the data types in the message, especially when the recipient is external to the company. If the message is not authorized, the system sanitizes the parts of the message that are not allowed to be passed to the recipient or it simply deletes the message. Finally, the good data elements are transferred to database systems for archiving for historical and trend analysis and then they are passed to intranet display. The data archived are viewable and accessible by someone with appropriate privileges. A logo and summary for the functions supported by trust systems is depicted in Figure 2.

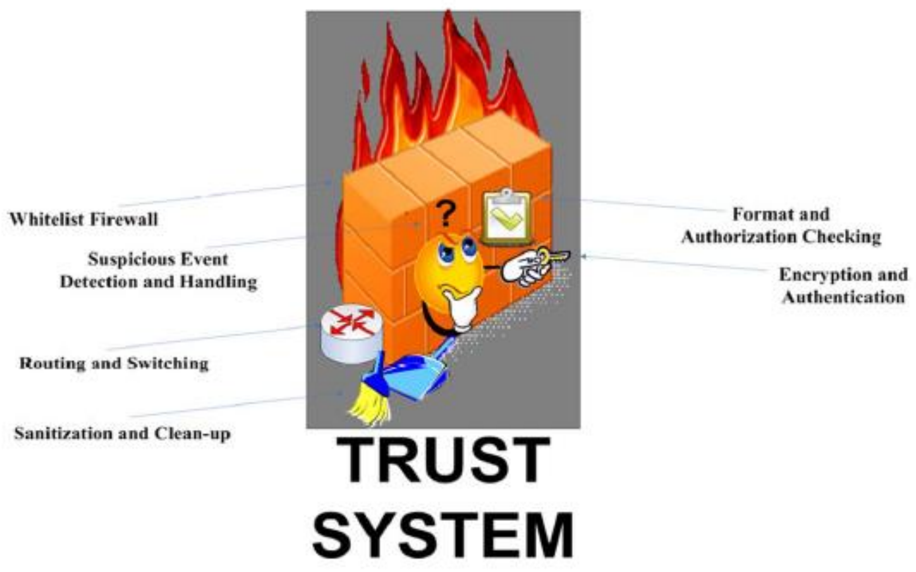

Whitelist Firewall
Suspicious Event Detection and Handling
Routing and Switching
Sanitization and Clean-up
Format and Authorization Checking
Encryption and Authentication
TRUST SYSTEM

Figure 2: Trust system logo with capabilities summary [2].

In [2], four different types of trust system implementation were discussed. Type 1 is the passive mode, type 2 is the half active mode, type 3 is the active mode, type 4 is the tunnel (or gateway) mode. In our work, trust systems will be implemented in type 3, active mode.

In type 1, passive mode, a trust system is connected to the switch on the communication link from the outside of the network. The advantage of passive mode is that a failed trust system does not block the communications link because trust system connected to the communication links from outside. The disadvantage is that trust system can detect and report the malicious packets and it cannot block these malicious packets. In type 2, half-active mode, implementation of passive mode is updated to block the bad activities in such a way that trust system interacts with a separate firewall or router Access Control List (ACL). The advantage is that it monitors and controls the traffics from outside the networks and same as passive mode if trust system is failed the communication links does not block. The disadvantage is that there will still be some chance that one or more bad activities will reach to the destination. In type 3, active mode, trust system is in line with all of the communication between the SCADA network and it can block bad data. The advantage is that it can detect and block the malicious traffics. The disadvantage is that if trust system fails the communication links is blocked. In type 4, tunnel or gateway mode, trust systems or routers provide firewall and other security features for the nodes behind them. The communications between them are secured by the encryption gateway. The advantage is that traffic packets are encrypted and protected when they are traveling from outside of the network. The disadvantage is that encryption and decryption of the packets cause delays to the network. Figure 3 demonstrates these 4 types of trust system implementation.

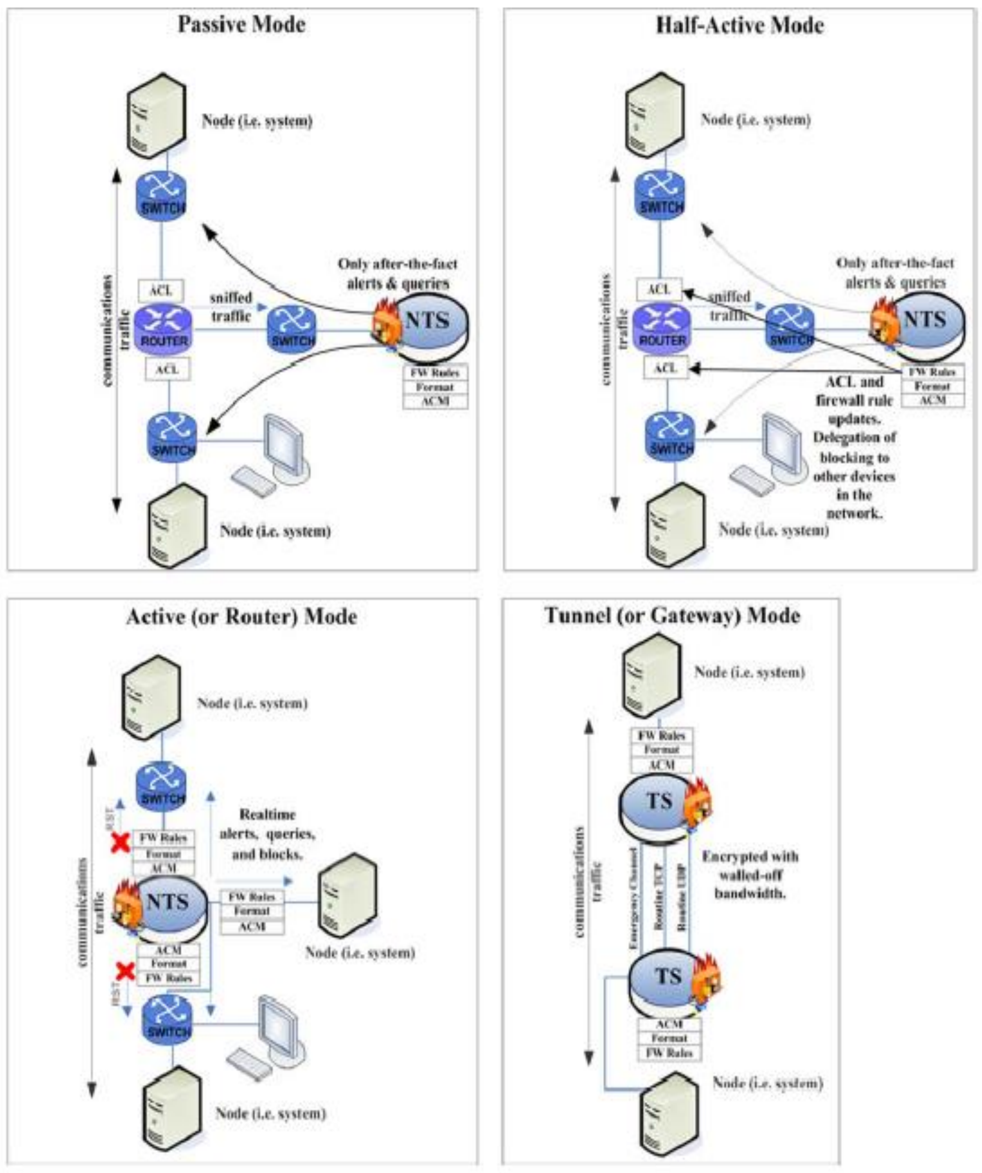


Figure 3: Trust system modes and configuration options [2]

In [6], a blackout reported that affected more than 50 million people in Italy. The Midwest independent system operator had only non-real-time data to work with, and they cannot identify the location of breaking lines.

In [7], a key management, wide area key exchange (WAKE), was employed to provide security for wide area measurement system (WAMS). The WAMS has four major hardware elements.

1. Phasor measurement units (PMUs), devices which measure the electrical waves on an electricity grid using a common time source for synchronization
2. Phasor data concentrators (PDCs)
3. Wide area network (WAN)
4. Real-time database and data archiver

To provide the security, WAKE uses public key infrastructure (PKI) [8]. PKI is an industry-standard asymmetric-key cryptosystem, with standards including X.509 and RFC 5280 [9].

In [10]- [12], SCADA networks are divided into small networks and trust systems are installed to the appropriate nodes to monitor and control the ingress and egress traffics. The links between segments are known as inter-segment links. Trust systems monitor the traffic between segments; consequently, intersegments shall be connected to at least one trust node which hosts trust system. Figure 4 demonstrates an example of segmentation. Inter-segment links are shown by the dotted lines. Trust systems are installed to the bordering nodes. Inter-segment links are connected to at least one trust node.

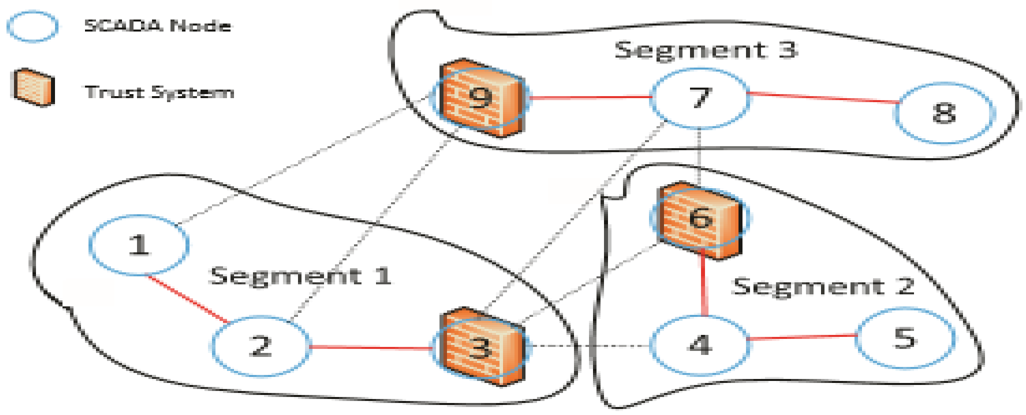


Figure 4: A simple example of a segmentation-based TSP11].

In [10], centrality measurement, method of ranking node in a graph, is utilized to improve the cyber protection of smart grid networks. The node degree, number of links connected to the node, is the simplest definition of centrality. Bordering nodes were considered for trust node placement.

In [11], the impact of latency was considered in selection of trust nodes because trust nodes distribute the time critical messages. In [11], the number of segments depends on the latency threshold.

In [12], link coverage and path tolerance deployment schemes were used to protect the SCADA networks against the cyber-attacks. Link coverage refers to the number of monitored links in a network and path tolerance refers to the maximum number of consecutive non-monitoring nodes in a route. As a consequence, the maximum link coverage and the least path tolerance were two ideal goals to protect the SCADA network against cyber-attack.

2.2 Description of the Method of Optimal Trust Nodes Placement from [1]

Table 1 describes symbols used in the algorithms, equations and formulas.

Table 1: Description of symbols [1]

Symbol	Description
G(V, E)	SCADA network graph. It is an undirected graph.
T(V, E^{MST})	MST of a given network G(V, E).
V	Node set of SCADA network.

E	Link set of the SCADA network.
N	Size of the SCADA network in terms of node, |V|=N.
E^{MST}	MST link set of the SCADA network.
α,β	Weighting factors for multiple objectives.
S	Set of network segments.
K	Total number of segments to be created, |S|=k.
e, s, b	Index variables for links, segments and bordering nodes respectively.
$e_{u\leftrightarrow v}$	Undirected link between node u and v.
d_u^{MST}, d_v^{MST}	Degrees of node u and v in the MST.
$d_{min}^{MST}(e_{u\leftrightarrow v})$	Minimum degree of the link between node u and node v in the MST.
w(e)	Weight of the link e in terms of propagation delay.
$\widetilde{w}(e)$	Normalized weight of the link e with respect to the maximum link weight.
$X_{\mathrm{I}}(l)$	Variable set for bordering nodes belonging to the inter-segment link l.
$L_{ss'}$	Set of inter-segment links between segments S, S'.
B(s)	Set of bordering nodes in the segment s.
Q	Total number of trust systems required
M	Number of available trust systems.
Y	Vector for binary decision variables for MST link elimination, Y=$(y_e)_{(N-1)\times 1}$
X	Vector for binary decision variables for trust node selection, X=$(x_{sb})_{\sum_{s\in S}\|B(s)\|\times 1}$.

The section has the following structure. Algorithm of sorting (Bubble sort) is considered in Section 2.2.1. Bubble sort will be used for Disjoint-set algorithm in Section 2.2.2. Disjoint-set algorithms are considered in section 2.2.2. It will be used for initial tree partitioning problem in Section 2.3.1. Kruskal algorithm of minimum spanning tree construction is considered in Section 2.2.3. It is necessary for initial tree partitioning problem in Section 2.3.1.

2.2.1 Sorting Algorithms

As mentioned, segmentation is based on the MST graph and sorting algorithm is the main part of the MST graph algorithm. In [13], types of sorting algorithms are enlisted. In this section, Bubble sort algorithm [14] will be described.

Bubble sort algorithm starts at the beginning of the input array, A[0,,n-1] of data to be sorted, and compares two neighboring elements. If the first one is greater than the second one, it swaps them and continues doing this to the end of the array. Again, and again, it starts from the beginning of the array and compares each pair of the neighboring elements until there are no swaps. Bubble sort algorithm pseudocode is as follows:

Algorithm 2.1: Pseudocode of bubble sort algorithm [14].

Bubble sort

```
1.    Input: An unsorted array of numbers,  A
2.    Output: The sorted in increasing order array, A
3.    Procedure BubbleSort ( A)
4.          n=length(A); swapped=true;
5.          While (swapped ==true)
6.            swapped=false;
7.            for i=1 to n-1 do
```

```
8.                  if A[i-1]>A[i] then
9.                      swap(A[i-1],A[i])
10.                     swapped=true
11.                 end if
12.             end for
13.         end while
14.     end procedure
```

The input of the Algorithm 2.1 is the array of numbers that are not sorted, and the output is the array of numbers sort from lowest number to greatest number. In Line 4 the variable **n** denotes the length of the array. Variable **swapped** locates in the **while** loop (Lines 5-13) and the initial value of **swapped** is **true**. In lines 7-12 if the condition in line 8 is true then the greatest value is swapped with the lowest value in the pair (Line 9) and then the value of the **swapped** is changed to true. Lines 7-12 repeat **n-1** times, where **n** is the length of the array. When the **for** loop is finished, the value of the **swapped** is false if there were no swaps in the array (Lines 7-12), and the **while** loop terminates (Line 5). After termination of the **while** loop, the numbers in the array are sorted from lowest to the greatest value. An example of the bubble sort work is given in Example 1.

Example 1. Application of Bubblesort to array A[5]= (5,1,4,2,8)

Input: Array of numbers: A[5]= (5,1,4,2,8)

n=5, length of the array

swapped=true;

(First iteration of **while** loop)

swapped=false; i=1;

(**5**, **1,** 4, 2, 8) → (**1, 5,** 4, 2, 8), here, algorithm compares the first two elements, and swaps since 5 > 1.

swapped=true; i=2;

(1, **5, 4,** 2, 8) → (1, **4, 5,** 2, 8), swap since 5 > 4

Swapped=true; i=3;

(1, 4, **5, 2,** 8) → (1, 4, **2, 5,** 8), swap since 5 > 2

Swapped=true; i=4

(1, 4, 2, **5, 8**) → (1, 4, 2, **5, 8**), now, since these elements are already in order (8 > 5), algorithm does not swap them.

Loop on I terminates

(Second iteration of **while** loop)

Swapped=false; i=1;

(**1, 4,** 2, 5, 8) → (**1, 4,** 2, 5, 8,)

i=2;

(1, **4, 2,** 5, 8) → (1, **2, 4,** 5, 8), swap since 4 > 2

Swapped=true; i=3;

(1, 2, **4, 5,** 8) → (1, 2, **4, 5,** 8)

i=4;

(1, 2, 4, **5, 8**) → (1, 2, 4, **5, 8)**

Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one iteration of **while** loop without **any** swap to know it is sorted.

(Third iteration of **while** loop**)**

Swapped=false; i=1;

(**1, 2,** 4, 5, 8) → (**1, 2,** 4, 5, 8)

i=2;

(1, **2, 4,** 5, 8) → (1, **2, 4,** 5, 8)

i=3;

(1, 2, **4, 5,** 8) → (1, 2, **4, 5,** 8)

i=4;

(1, 2, 4, **5, 8**) → (1, 2, 4, **5, 8**)

The value of the swapped does not change, hence, the iteration of **while** loop is terminated, and the array is sorted.

Output: A[5]=(1, 2, 4, 5, 8).

2.2.2 Disjoint-Set Algorithms

Before going to describe the Disjoint-set algorithm, some definitions of the graph need to be cleared. Figure 5 illustrates an example of small graph. A graph, G, is a pair of sets, G(V, E), where V is set of vertices, G.V={1, 2, 3, 4, 5, 6, 7}, E is a set of edges, G.E={(1,2), (2,3), (3,5), {4,5}, (5,6), (6,7)}. And edge, e, is a pair of vertices, (a, b), that these vertices of the edge are known as end-vertices of the edge. A graph in Figure 5 is a weighted graph, each edge has a weight. Weight of the edge is denoted by w and the set of edge with weight is denoted by G.E.w={((1,2),4), ((2,3),10),((3,5),6), ((4,5),3),((5,6),4),((6,7),1)}.

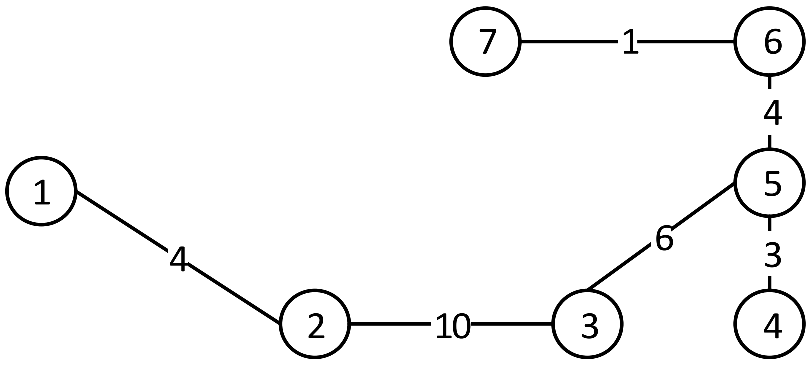


Figure 5: An example of graph [1].

The number of vertices defines by |G.V| and the number of edges of the graph, G, defines by |G.E|. In Figure 5 the number of vertices is 7, |G.V|=7, and the number of edges is 6, |G.E|=6.

Disjoint-set algorithm [15] is used to find the connected vertices of a graph and creates a set of minimum spanning trees of a graph [16]. It plays a key role in Kruskal algorithm [17] for finding the minimum spanning trees of a graph. Parent pointer and rank value are the attributes of the elements of the disjoint-set forest. If the parent pointer of an element does not point to another element, then the element is the root of the tree and is the representative member of its set. If the parent pointer of the element points to another element, it means that this element belongs to the set, tree, and the set is identified the chain of parents upward until a representative element is reached at the root of the tree. Figure 6 illustrates an example to show the parent pointer.

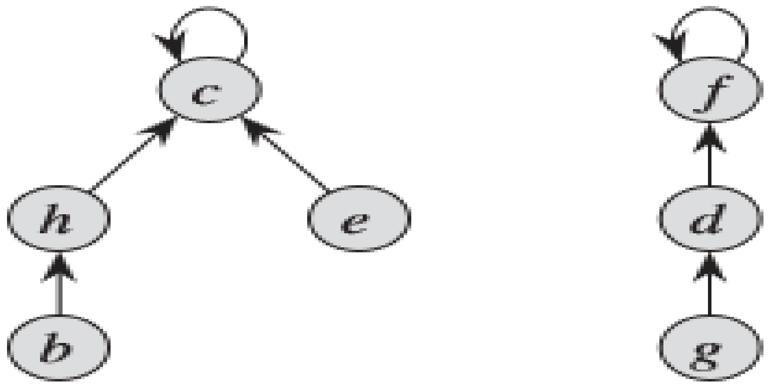


Figure 6: An example of parent pointer definition in graph.

In Figure 6, vertices C and F point to themselves. It means that they are the root of the trees. The vertex b points to another vertex, it means that this vertex belong to the tree, and this vertex is not the root, it points to the vertex c. As a consequence, to find the tree that vertex b is belong to, vertex b follows a chain to up to find the vertex that points to itself, root of the tree. In Figure 6, the set of left tree is {b,c,e,h} and the set of right tree is {d,f,g}. Another attribute is rank which denotes the depth of the tree. For example, both trees in Figure 6 have a rank of 2.

Disjoint-set algorithm contains three functions [15]. First one is **Make-set(x)** function. The input of the **Make-Set(x)** is a vertex of the graph. This function creates a new set for the input vertex and initialize the attributes of the vertex (parent pointer points to itself and the value of the rank is 0). This function locates in a **for** loop to create a new set for all vertices in a graph. Algorithm 2.1 shows the pseudocode of the **Make-Set** function.

Algorithm 2.2: Pseudo code of the Make-Set function [15].

Make-Set

```
1.  Input: x, vertex of graph.
2.  Output: a set of disjoint-set tree with the initialized attributes.
3.  Function Make-Set (x)
4.  Begin
5.      x.parent =x;
6.      x.rank=0;
7.  End
```

In Line 5, vertex x points to itself, create a tree that the root of the tree is itself, and the rank value in Line 6 is 0 that means the depth of the tree is 0. As a result, a new set for the x is created in the disjoint-set tree and the attributes of the vertex are initialized.

The second function is **Find(x)**. The input of the Find function is a vertex of the tree. The functionality of this function is to determine which set of the disjoint-set tree contains a given vertex **x**. Algorithm 2.3 describes the pseudocode of the **Find(x)**.

Algorithm 2.3: Pseudo code of the Find(x) function [15].

```
Find
1.    Input: vertex x
2.    Output: root of vertex x.
3.    Function Find(x)
4.    Begin
5.    If x.parent !=x then
6.        x.parent:=Find(x.parent)
7.    End if
8.    Return x.parent
```

The input of the function is a vertex of the graph. If the parent pointer of the vertex x doesn't point to itself (if point to itself, it means this is the root vertex) (Line 5) then it goes up the tree till find the root vertex (Line 6). The output is the root of the input vertex. Hence, the set of vertex x is identified because each set have only one specific root.

The third function is **Union (x, y)**. This function uses the **Find** function for vertex **x** and **y** of the edge **(x, y)** to find the roots of trees that they belong to them. If the roots of vertices **x** and **y** are different then the trees are combined by attaching the root of one to the root of the other. This union function uses the **rank** value which means that the shorter tree will be attached to the root of the taller tree. Algorithm 2.7 shows the pseudocode of the Union (x, y) algorithm.

Algorithm 2.4: Pseudo code of the Union function [15].

```
Union
1.   Input: Edge of the graph
2.   Output: Updating the structure of trees
3.   Function Union(x, y)
4.   Begin
5.     xRoot:=Find(x)
6.     yRoot:=Find(y)
7.     If   xRoot == yRoot then
8.           Return
9.     End If
10.    If   xRoot.rank < yRoot.rank then
11.           xRoot, yRoot:=yRoot, xRoot. //swap xRoot and yRoot
12.    yRoot.parent := xRoot
13.  End If
14.    If   xRoot.rank==yRoot.rank then
15.           xRoot.rank:= xRoot.rank+1
16.  End If
17.  End
```

Lines (5-6) use Find function to find the root of trees (xRoot and yRoot) that vertices x and y are belong to them. If the roots of trees are same, then vertex **x** and **y** are in a same set. If the roots of vertex x and y are different and the **rank** value of the vertex

x is lower than the rank value of the vertex **y** then the value of the ranks for the vertex x and y will be swapped (Lines 10-13). If their rank value of **x** and **y** are same then, the rank value of the vertex x will be increase by 1 unit (Lines 14-16).

Algorithm 2.5 shows the pseudocode of Disjoint-set algorithm.

Algorithm 2.5: Pseudocode of Disjoint-set algorithm.

Disjoint-set

```
1.  Input: Edges of the Graph, G, and N number of vertices in the graph,
    |G.V|
2.  Output: Collection of disjoint sets
3.  Function Disjoint-set(G.E, N)
4.  Begin
5.  N=|G.V|;
6.      For i=1 to N| do
7.          Make-Set(Vi);
8.      End For
9.      For j=1 to |G.E| do
10.         X=Find(Ej. e1)// first vertex of the edge Ej;
11.         Y=Find(Ej. e2)// second vertex of the edge Ej;
12.         Union(X,Y)
13.     End For
14. End
```

The input of the Disjoint-set algorithm is a graph and the output is the set of trees. Lines 6-7 executes the Make-Set function for each vertex of the graph, Vi, where V is the vertex and the index i denotes the vertex number from the set of the vertex of the graph, G.V={V1,V2,…..,VN} and N is the number of vertexes in a graph, N=|G.V| . Lines 8-12 is a for loop to check all the edges of the graph. In Lines 10-11 the Find function is called to find the parent of the endpoints of the edge, (Ej.e1,Ej.e2), and store in variables X and Y. in Line 12 the function Union(X, Y) is called to find the roots of the trees. Figure 7 shows the four connected partitions of a graph that is used as an input for Disjoint-set algorithm. An example of Disjoint-set work is given in Example 2.

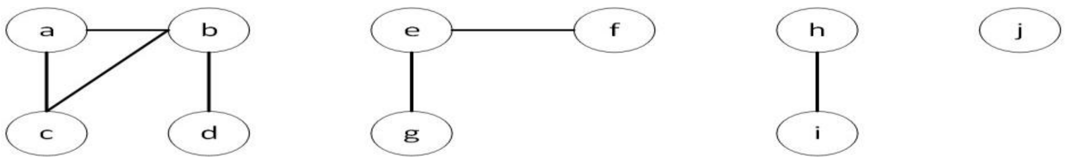


Figure 7: Four connected components of a graph [15].

Example 2: Application of Disjoint-set to the graph G;

Input: Four connected graph G(V,E): G.V={a,b,c,d,e,f,g,h,I,j}, G.E={(a,b),(a,c),(c,b), (b,d),(e,f),(e,g),(h,i)}, |G.v|=10, Number of vertices, |G.E|=7, Number of edges.

Starting the first loop to create the sets for each vertex in given graph

i=1;

Make-Set(a)={a}, here Make-Set algorithm create a set for vertex a.

i=2;

Make-Set(b)={b}, here Make-Set algorithm create a set for vertex b.

i=3;

Make-Set(c)={c}, here Make-Set algorithm create a set for vertex c.

i=4;

Make-Set(d)={d}, here Make-Set algorithm create a set for vertex d.

i=5;

Make-Set(e)={e}, here Make-Set algorithm create a set for vertex e.

i=6;

Make-Set(f)={f}, here Make-Set algorithm create a set for vertex f.

i=7;

Make-Set(g)={g}, here Make-Set algorithm create a set for vertex g.

i=8;

Make-Set(h)={h}, here Make-Set algorithm create a set for vertex h.

i=9;

Make-Set(i)={i}, here Make-Set algorithm create a set for vertex i.

i=10;

Make-Set(j)={j}, here Make-Set algorithm create a set for vertex j.

The first for loop is finished.

Starting the second for loop. This loop iterates for the size of the edge, |G.E|=7.

j=1;

X=Find(a); //X={a};

Y=Find(b);// Y={b};

Union(a,b);// sets {a},{b} are combined and create one set, {a,b};

j=2;

X=Find(a); //X={a,b};

Y=Find(c);// Y={c};

Union(a,c);// sets {a,b},{c} are combined and create one set, {a,b,c};

j=3;

X=Find(c); //X={a,b,c};

```
Y=Find(b);// Y={a,b,c};
Union(c,b);// they are in same set;
j=4;
X=Find(b); //X={a,b,c};
Y=Find(d);// Y={d};
Union(b,d);// sets {a,b,c},{d} are combined and create one set, {a,b,c,d};
j=5;
X=Find(e); //X={e};
Y=Find(f);// Y={f};
Union(e,f);// sets {e},{f} are combined and create one set, {e,f};
j=6;
X=Find(e); //X={e,f};
Y=Find(g);// Y={g};
Union(e,g);// sets {e,f},{g} are combined and create one set, {e,f,g};
j=7;
X=Find(h); //X={h};
Y=Find(i);// Y={i};
Union(h,i);// sets {h},{i} are combined and create one set, {h,i};
End of second for loop;
Output: {a,b,c,d}, {e,f,g}, {h,i};
```

2.2.3 Kruskal Algorithm of MST Construction

Given a weighted graph, MST is a graph connecting all the vertices of the graph without any cycle between the vertices and also with minimum possible edge weight [16]. In [17], the Kruskal algorithm is used to obtain the MST graph. Pseudocode below presents the Kruskal algorithm.

Algorithm 2.6: Pseudo code of Kruskal algorithm [17].

```
MST-Kruskal (G(V,E))
1.  Input: An undirected graph having vertices, G.V={V1,V2,….,VN},
    and edges, G,E, with weights G.E.w for each e from
    G.E={G.E1,G.E2…..G.EM}.
2.  Output: An undirected graph without any cycle between vertices
    (Minimum Spanning Tree).
3.  A=∅; // A is a minimum spanning tree; N=|G.V|;// Number of vertices
4.  For i=1 to N do
5.      Make-Set(Vi);
6.  End For
7.  BubbleSort(G.E.w); //Sort the edges of G.E into non-decreasing order
    by weight w;
8.  For j=1 to |G.E| do// |G.E| denotes the number of edges in graph,G;
9.  If Find-Set(Ej.e1) ≠ Find-Set(Ej.e2) // Ej denotes the edge in graph in
    position j; e1 and e2 are endpoints of the edge
10.         A=A ∪ {G.Ej};
11.         Union (u, v);
12. Return A;
13. End If
14. End For
```

Input of the Kruskal algorithm is an undirected graph and the output is a minimum spanning tree. Line 3 initialize an empty set of minimum spanning tree and N, is the number of vertices. In line 5, **Make-set**, Algorithm 2.2, function creates a set of trees. In this set, all vertices in the input graph is a separate tree. Line 7 executes the BubbleSort function, Algorithm 2.1, to sort the set of edge into increasing order by weight of the edge. In line 8, first element from the set of the edge is selected. The functionality of **Find-Set(u)** and **Find-Set(v)** is to find the set of trees that contain vertices **u** and **v** of the edge (u, v), Algorithm 2.3. If selected edge connects two different trees (Line 9), their parents are different, then adds this edge to the minimum spanning tree set **A** (Line 10)**.** In Line 10, **union (u, v)** combines two trees that contain vertices **u** and **v**. Line 12 returns the minimum spanning tree of the input graph.

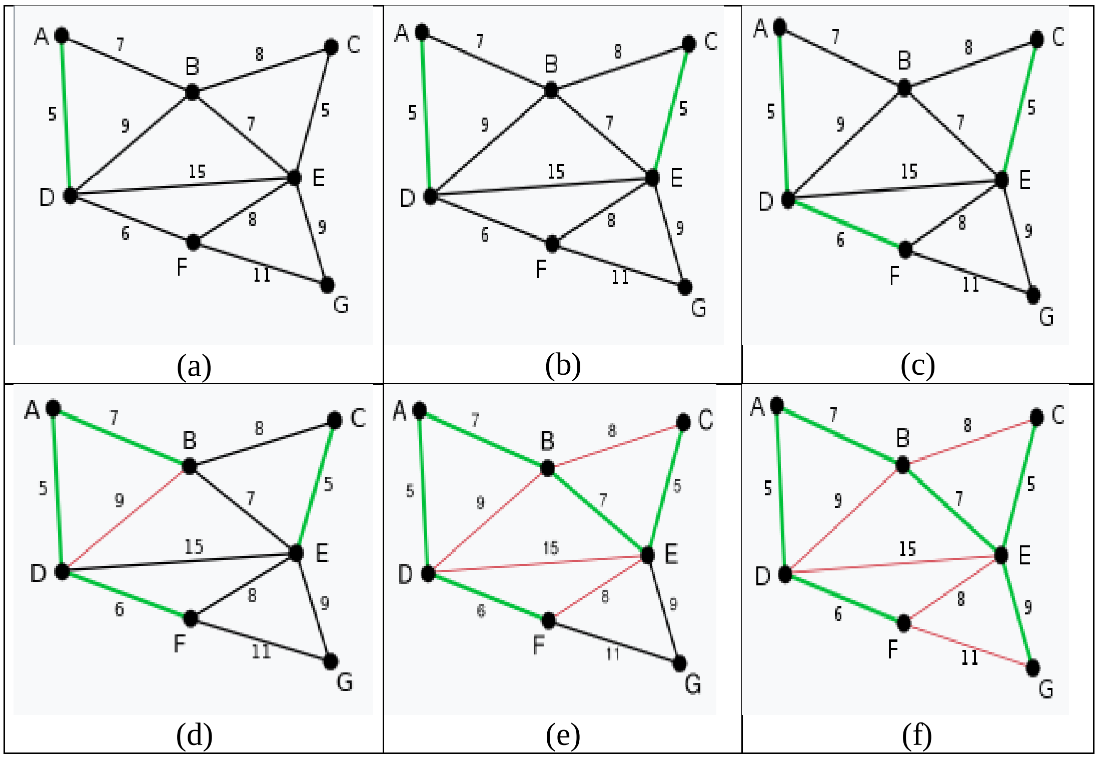


Figure 8: An illustrative example of Kruskal algorithm. a) edge **AD** removed from input graph and added to minimum spanning tree. b) edge **CE** removed and added to minimum spanning tree .c) edge **DF** removed and added to minimum spanning tree. d)edge **AB** added to minimum spanning tree, edge **BD** form a cycle. e) edge **BE** added to MST. f) last edge **EG** is removed from input graph and added to MST [17].

Figure 8(a) shows the input graph. The edge **AD** with minimum weight is selected to remove from input graph and add to the minimum spanning tree (MST), it is shown by the green line. In Figure 8(b), the next edge **CE** with minimum weight is removed from the input graph and added to the MST. Figure 8(c) shows the next selected edge **AB**. Edge **BD** cannot be added to the MST because it forms a cycle **ADB**. The edges with red line demonstrate the cycle, hence, they cannot be added to MST. Finally, the MST is shown by green line in Figure 8(f). An example of MST-Kruskal work is given in Example 3.

Example 2: Application of MST-Kruskal to the graph G;

Input: Graph G.// G.V={A,B,C,D,E,F,G};

G.E.w={((A,B),7),((B,C),8),((A,D),5),((B,D),9),((B,E),7),((C,E),5),((D,E),15),((D,F),6),((F,G),11),((E,F),8),((E,G),9)}; N=|G.V|=7, number of vertices; |G.E|=11 is number of edges

Starting the first for loop to create the sets for each vertex in given graph

i=1;

Make-Set(A)={A}, here Make-Set algorithm create a set for vertex A.

i=2;

Make-Set(B)={B}, here Make-Set algorithm create a set for vertex B.

i=3;

Make-Set(C)={C}, here Make-Set algorithm create a set for vertex C.

i=4;

Make-Set(D)={D}, here Make-Set algorithm create a set for vertex D.

i=5;

Make-Set(E)={E}, here Make-Set algorithm create a set for vertex E.

i=6;

Make-Set(F)={F}, here Make-Set algorithm create a set for vertex F.

i=7;

Make-Set(G)={G}, here Make-Set algorithm create a set for vertex G.

End of first for loop

BubbleSort(G.E.w)={((A,D),5),((C,E),5),((D,F),6),((A,B),7),((B,E),7),((B,C),8),((E,F),8),((B,D),9),((E,G),9),((F,G),11),((E,D),15)}

Starting of second for loop// it iterates 11 times for the edges of the graph G

j=1;

Find(A)≠Find(D)

A.E.w={((A,D),5)}

Union(A,D);// sets {A} and {D} are combined,{A,D};

j=2;

Find(C)≠Find(E)

A.E.w={((A,D),5),((C,E),5)}

Union(C,E);// sets {C} and {E} are combined,{C,E};

j=3;

Find(D)≠Find(F)

A.E.w={((A,D),5), ((C,E),5), ((D,F),6)}

Union(D,F);// sets {A,D} and {F} are combined,{A,D,F};

j=4;

Find(A)≠Find(B)

A.E.w={((A,D),5), ((C,E),5), ((D,F),6), ((A,B),7)}

Union(A,B);// sets {A,D,F} and {B} are combined,{A,D,F,B};

j=5;

Find(B)≠Find(E)

A.E.w={((A,D),5) , ((C,E),5), ((D,F),6), ((A,B),7),((B,E),7)}

Union(B,E);// sets {A,D,F,B} and {C,E} are combined,{A,B,C,D,E,F};

j=6;

Find(B)=Find(C)// they are in same set, and cannot be added to A because they make a cycle

j=7;

Find(E)=Find(F)// they are in same set, and cannot be added to A because they make cycle

j=8;

Find(B)=Find(D)// they are in same set, and cannot be added to A because they make cycle

j=9;

Find(E)≠Find(G)

A.E.w={((A,D),5),((C,E),5), ((D,F),6), ((A,B),7),((B,E),7), ((E,G),9)}

Union(E,G);// sets {A,B,C,D,E,F} and {G} are combined,{A,B,C,D,E,F,G};

j=10;

Find(F)=Find(G) // they are in same set, and cannot be added to A because they make cycle

j=11;

Find(E)=Find(D) // they are in same set, and cannot be added to A because they make cycle

Output: A={((A,D),5),((C,E),5), ((D,F),6), ((A,B),7),((B,E),7), ((E,G),9)}

2.2.4 Linear Programming Problem (LPP) in General

Linear programming is a method to achieve the best outcome (such as maximum profit or lowest cost) with the special conditions and with certain restrictions. Linear programming is a special case of mathematical programming (also known as the mathematical optimization). In [18,19], linear programming problem is discussed. In our application, we used Matlab program to solve the LPP. Linear programming problem can be expressed as below.

$$\underset{x}{\text{optimize}} \, . \, C^T x \tag{2.1}$$

subject to

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n < b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n < b_2 \\ . \\ . \\ . \\ a_{m1}x_1 + a_{m2}x_2 + \cdots .. + a_{mn}x_n < b_m \end{array} \tag{2.2}$$

$$x_i \in \{0,1\} \tag{2.3}$$

where **x** is the vector of variables, the value of the variable must be 0 or 1, that must be determined, **C** and **b** are vectors of (known) coefficients, and $(.)^T$ is the matrix transpose, and A is an m×n matrix of real numbers, a. (2.1) is called objective function that can be minimized or maximized. The inequalities (2.2) and (2.3) are the constraints which specify a convex polytope over which the objective function, f(x), is to be optimized. The output of the linear programming problem is a vector, x, that optimize, minimize or maximize, the given objective function, f(x).

2.3 Definition of Optimal TSP Problem as LPP

In [1], an electric power grid system is considered as an undirected graph, in which nodes correspond to power grid buses, generator or load, and links correspond to the power grid branches, transformer or transmission lines, and the weight of the links denotes the propagation delay or distance between two endpoints of the link. Figure 9 shows an example of a power grid system with 9 power grid buses and 12 power grid branches.

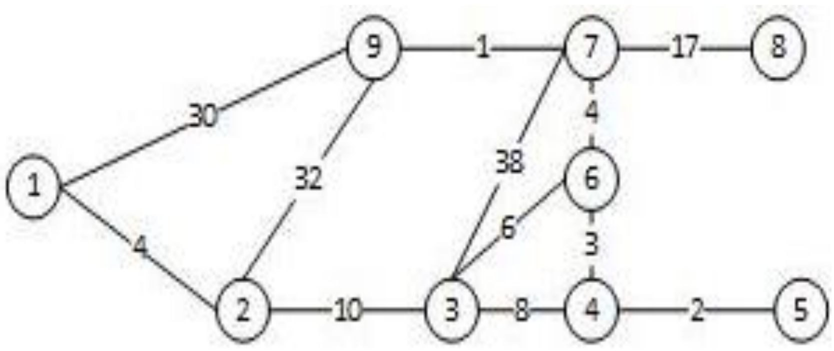


Figure 9: A graph G(V,E) of power grid system. with G,V={V1...V9} power grid buses (nodes) and G.E={G.E1,..,G.E12} power grid branches (links) [1]. Each edge has weight, propagation delay between two endpoints of the link or distance between them, marking it, e.g., w(G.E1(v1,v2))=4.

The definition of optimal TSP problem is divided into 3 parts presented in Sections 2.3.1-2.3.3. Section 2.3.1 presents a problem of segmentation to initialize the partitions, Section 2.3.2 presents a problem of local search to update the partitions optimally, and Section 2.3.3 describes the problem of optimal trust node selection from the bordering nodes of the partitions.

Figures 10-14 illustrate the results of the initial tree partitioning, ideal segmentation, and trust node selection problems solving for SCADA network from Figure 9.

2.3.1 Definition of Initial Tree Partitioning Problem (Segmentation)

As I mentioned in the introduction, SCADA networks are distributed geographically, the distance between nodes in SCADA networks maybe hundreds or thousands of miles. The initial tree partitioning problem (Segmentation) is used to divide the SCADA networks to segments. MST is the main part of the segmentation. The MST of a SCADA network is obtained by the Kruskal algorithm, Algorithm 2.6. The partitions of the SCADA networks include the nodes that the distance between them is minimal because partitions are in MST form.

Linear programing problem (LPP1) is used to construct the primary partitions by removing links from the SCADA network, G(V,E). By removing **K-1** links from the SCADA network, this network will be divided into **K** segments. The MST of the SCADA network contains N-1 links, where N denotes the quantity of SCADA nodes, |G.V|. Totally, N-K edges must be left from the SCADA network to achieve the K partitions.

LPP1 is initial tree partitioning that eliminates K-1 MST links to obtain segments regarding their normalized weights and lowest degrees. Node's degree is the number of links connected to that node. Leaf nodes degree is one. The minimum degree of the link is defined as follows [1]:

$$\mathrm{d}_{\mathrm{min}}^{\mathrm{MST}}(\mathrm{e}_{\mathrm{u}\leftrightarrow\mathrm{v}}) = \min(\mathrm{d}_{\mathrm{u}}^{\mathrm{MST}}, \mathrm{d}_{\mathrm{v}}^{\mathrm{MST}}) \ , \tag{2.4}$$

where d_u^{MST} is the degree of the node **u,** that is, the number of links that are connected to the node **u**. In (2.4), the lowest degree of the edge of the MST graph is obtained by selecting the minimum degree of the nodes that are connected to that link. Figure 10 depicts by rectangles the minimum degree of the links of the SCADA network in Figure 9.

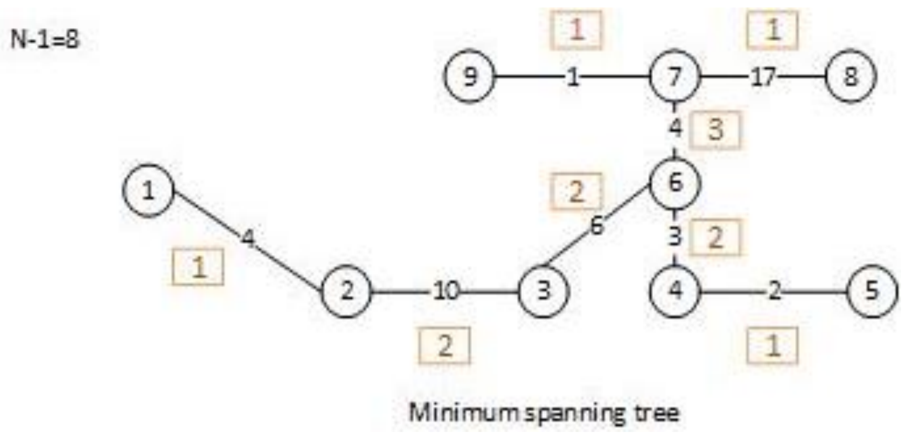


Figure 10: Minimum spanning tree of SCADA network of figure 9 [1].

In LPP1, two weighting factors (α, β) are normally set to α=1 and β=0.5. Normalized weight (2.7), sets the weight of the links in a same range when they are not in the same range. For example, in Figure 10 the link (7, 8) has the maximum value in the MST SCADA network and it is equal to 17. To obtain the normalized weights, weights of all the links are divided by 17. The normalized weight of link (1, 2) is equal to $\frac{4}{17} = 0.2352$. The decision variable in LPP1 is, Y=$(y_e)_{(N-1)\times 1}$, such that [1]:

$$y_e = \begin{cases} 1, & \text{if } e \in E^{MST} \text{ is selected for elimination;} \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

Initial tree partitioning problem (LPP1) is described as follows [1]:

$$\max_{Y} \sum_{e \in E^{MST}} \left(\alpha d_{\min}^{MST}(e) + \beta \widetilde{w}(e) \right) y_e \quad , \tag{2.6}$$

where

$$\widetilde{w}(e) = \frac{w(e)}{\max_{e \in E} w(e)}, \forall e \in E \quad , \tag{2.7}$$

subject to

$$\sum_{e \in E^{MST}} y_e = K - 1 \quad , \tag{2.8}$$

$$y_e - d_{min}^{MST}(e) < 0, \ \forall e \in E^{MST} \quad , \tag{2.9}$$

$$y_e \in \{0,1\}, \quad \forall e \in E^{MST} \quad , \tag{2.10}$$

The objective function is presented in (2.6) and the normalization weight is defined in (2.7). Constraint (2.8) is the quantity of the links that must be removed, and this is equal to K-1 which means that if the quantity of segments K is equal to 3 the number of eliminated links shall be equal to 2. Constraint (2.9) ensures that each segment has at least two nodes that are connected. Algorithm 2.7 gives the pseudo code of the initial tree partitioning problem [1].

Algorithm 2.7: Pseudocode of initial tree partitioning (segmentation) algorithm [1].

```
Initial tree partitioning
Input: G (V, E), SCADA network, with weights, G.E.w; K, number of target segments;
Output: S= {s1, s2, s3, …, sK}, K segments obtained;
1.  Begin
2.  $E^{ss} = \emptyset, N = |G.V|, k_s = \left\lceil \frac{N}{K} \right\rceil$; // N is number of nodes in graph G; $E^{ss}$
    is the set of links left after eliminating N-K links; $k_s$ is a limitation
    variable that the size of segments must not exceed from this variable.
3.  $T(V, E^{MST}) \leftarrow MST - \mathbf{Kruskal}(G(V, E))$; // MST graph, T, is
    calculated by the Algorithm 2.6.
4.  For all e$\in E^{MST}$ do
```

```
5.      Calculate the lowest degree $d_{\min}^{MST}(e)$ by (2,4);
6.      Calculate the normalized weight $\tilde{w}(e)$ by (2.7);
7.    End for
8.    $E^{I} \leftarrow$
      $\mathbf{LPP1}(\mathbf{inputs}\text{: objective function, (2.6), and the constraints, (2.8)} -$
      $\text{(2.10);}\quad \mathbf{Output}\text{: the vectore Y that maximize the objective function))}$
      (2.5)-(2.10);   // $E^{I}$, vector Y, is the set of links that need to be
      eliminated. LPP1 will choose the (K-1) edges that shall be removed to
      make the primary tree partition.
9.    $E^{SS} = \{E^{MST} \backslash E^{I}\}$; // Remove $E^{I}$from the links of MST and store the
      remain links in $E^{SS}$. Size of $E^{SS}$ is N-K.
10.   $\mathbf{return}\ S = \{s1, s2, \ldots., sK\} \leftarrow \mathbf{Disjoint-set}(\mathbf{E^{SS}}, \mathbf{N})$;   //initial
      partition set defined by using Disjoint-set algorithm, Algorithm 2.5.
      Inputs of Disjoint-set algorithm are edges, $E^{SS}$, and number of nodes
      in the graph G, N.
```

The input of the Algorithm 2.7 is the graph G (V, E) which G.V and G.E, vertices and edges of G, and G.E.w, weights of the edges, G.E, and the number of segments, K. The output of the Algorithm 2.7 is the set of the segments. Line 2 is the initialization part and the set of the links left after elimination (E^{ss}) by the LPP1 is empty, number of nodes (N) set to the number of nodes in the input graph and variable $k_s = \left\lceil \frac{N}{K} \right\rceil$. In fact, the variable k_s is the limitation variable and it means that the number of nodes in segments must not exceed this value. In line 3, the minimum spanning tree, T(V, E^{MST}) is computed by the Kruskal algorithm. The input of the Kruskal algorithm is the original graph, G (V, E), and edges weights, and the output of the Kruskal algorithm is the MST graph, T(V, E^{MST}). The number of links in the MST is equal to N-1. Lines 4-6 compute the minimum degree and normalized weight for all links in the MST. K-1 links are eliminated by the LPP1 in line 8 and the set of the eliminated links of the MST is returned, E^{I}, by the LPP1. In line 9, the remaining set of links after elimination are placed in E^{ss}, set of links after elimination. Line 10 returns the initial set of segments. Disjoint-set algorithm, Algorithm 2.5, identifies the segments. The inputs of the Disjoint-Set algorithm are set of edges and number of nodes in the graph, G, and the output is the collection of disjoint sets, S. Figure 11 illustrates the primary segments that obtained by the Algorithm 2.7.

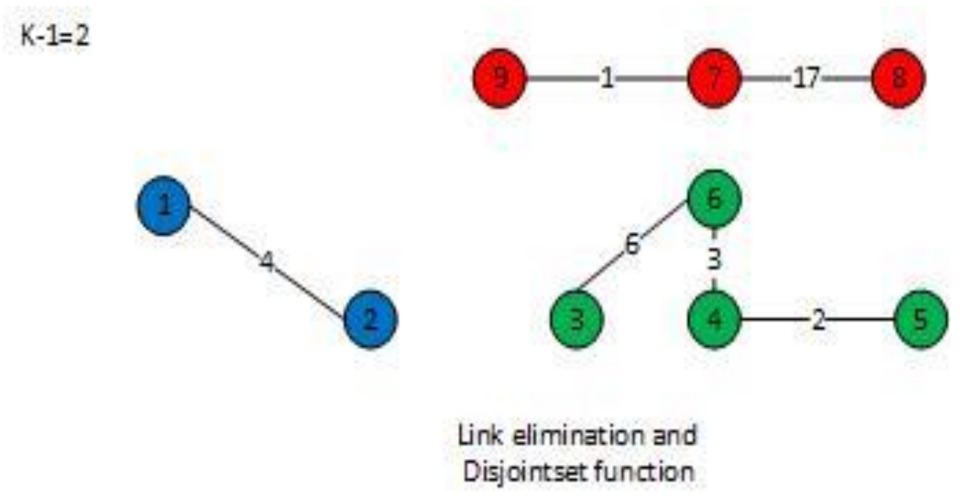


Figure 11: Links eliminations and segments identification [1].

The inputs of the initial tree partitioning algorithm, Algorithm 2.7, are the weighted graph, G, in Figure 9 and the number of target segments, K, which is 3. The variables E^{ss}, set of remain links after eliminating the links, number of nodes, N=|G.V|=9, and $k_s = 3$, limitation variable, are initialized. In line 3 the MST, T, is obtained by the MST-Kruskal algorithm. The MST is represented by Figure 10. Lines 4-7 compute the minimum degree of the links (2.4) and normalized the weight of the MST links (2.7). Figure 10 represents the minimum degree of the links. By solving the linear programming problem 1, LPP1, the number of links that must be eliminated is returned and stored in E^I, $E^I = \{(2,3),(6,7)\}$,. The input of LPP1 are objective function and the constraints (2.6)- (2.10). the output is the vector Y that the decision variable is denoted in (2.5). In line 9, set of the elimination links, E^I, is removed from the set of the MST links, E^{MST}, and the remain links of the MST are stored in E^{SS}, $E^{SS} = \{\big((1,2),4\big),\big((3,6),6\big),\big((6,4),3\big)\big((4,5),2\big),((9,7),1)((7,8),17)\}$. The set of 3 segments, S={s1,s2,s3}, is returned by Disjoint-Set algorithm, S= {{1,2},{3,4,5,6,},{7,8,9}}, Line 10.

2.3.2 Definition of Ideal Segmentation Problem

The objective of ideal segmentation problem is to reduce the sum of the minimum spanning tree weights of all partitions in SCADA networks and it is acquired when the MST weight of any partition is minimized. Ideal segmentation problem is described as follows [1]:

$$\min_{S} \sum_{s \in S} \tau_s^{mst} \ , \tag{2.11}$$

$$\text{where}$$

$$\tau_s^{mst} = \sum_{e \epsilon MST^s} w(e^s)\,, \forall s \ \in S \quad , \tag{2.12}$$

$$\text{subject to}$$

$$\bigcup_{s \in S} V^s = V\,, \tag{2.13}$$

$$V^s \cap \ V^{s'} = \emptyset, \forall s \neq s' \ \text{and}\ s, s' \ \in S\,, \tag{2.14}$$

$$|V^s| \leq \left\lceil \frac{N}{K} \right\rceil \ , \tag{2.15}$$

where V is set of the SCADA nodes, S is set of the segments; for each segment , $s \in S, \tau_s^{mst}$ is the MST's weight of the segment s and V^s is the set of segment's nodes, MST^s is the minimum spanning tree in segment s and $w(e^s)$denotes the link's weight in segment s. Expression (2.11) is the objective function that minimizes the sum of the weights of the MST graph in the segments. Equation (2.12) defines the sum of the weights of the MST graph in the segment s. Constraint (2.13) ensures that the entire network is segmented, i.e. the union of all segments returns the whole network. Constraint (2.14) shows that intersection of any two segments is empty, in other words, it shows that each node belongs to only one segment. Constraint (2.15) limits the number of nodes for one segment. Algorithm 2.8 describes the local search algorithm to provide the ideal segments size.

Algorithm 2.8: Pseudo code of local search algorithm.

Local search

Input: G(V, E), K, S // G is the original graph, K is number of target segments and S is the set of segments that returns by the Algorithm 2.7;

Output: Set of segment sets after repartitioning;

```
15. Begin
16.   Phi=0, N=|G.V|, $k_s=\left\lceil\frac{N}{K}\right\rceil$ ; Δmin =∅//initialization
17.   while phi < 1 do
18.    Count=0;
19.    For i=1 to K do
20.      For j=1 to K do
21.        If $|s_i>k_s$ and $|s_j|<k_s$ then
22.          Δmin=FindΔmin($s_i, s_i$, T) // The inputs of the FindΔmin
             algorithm,  Algorithm 2.9, are oversized segment, $s_i$, and
             undersized segment, $s_j$, and MST graph, T, the output is the
```

```
                minimum sub-segment, $\Delta\min$, belonging to $s_i$ that is adjacent to
                $s_j$.
23.             If (($|k_s - |s_i||$) + ($|k_s - |s_j||$) > (($|k_s - |s_i| + |\Delta\min||$) +
                ($|k_s - |s_j| - |\Delta\min||$)) then
24.             $s_i = \{s_i \setminus \Delta\min\}$ and $s_j = \{s_j \cup \Delta\min\}$;// Updates the MST
                partitions
                count=count+1;// It checks for balancing segment sizes
25.             End if
26.           End if
27.         End for //j
28.       End for //i
29.       If (count==0) then
30.         Phi=1;//termination condition
31.       End if
32.      End while
33.      Return s={s1,s2,…., sK}
34.    End
```

The inputs of the Algorithm 2.9 are complete graph of the SCADA network, G(V,E), number of segments, K, and the set of segment sets, S. Output contains updated segment set. In line 2 the value of k_s, $k_s = \left\lceil \frac{N}{K} \right\rceil$, is the limitation variable, ideal segmentation size (2.15), and it means that the number of nodes in segments must not exceed this value. The variable phi is set to zero, this variable is used by the while loop (Lines 3-20) implementing the local search. This loop searches for the oversized, s_i, ($|s_i| > k_s$) and the undersized, s_j, ($|s_j| < k_s$) segments, and for each pair of oversized and undersized segments that are adjacent, computes sub set, Δmin, of the oversized segment, Algorithm 2.9, (Line 9). Line 10 checks that the sum of sizes of oversized and undersized segments after updating (removing the subset, computed by Δmin algorithm, from the oversized segment, s_i, and adding this subset to the undersized segment, s_j) is less than it was originally. If the condition in line 10 is true, then the Δmin set is removed from the oversized segment and added to the undersized segment (Line 11). Δmin is computed in algorithm 2.9. This loop will be finished when the adjacent oversized and undersized segments are not exist, after that the value of the counting variable sets to one (phi=1). Line 20 returns the segment set.

Algorithm 2.9: Pseudo code of Δmin finding algorithm.

```
Find Δmin
Input: $s_i, s_j$ and T(V, $E^{MST}$),// $s_i$ is the oversized segment set, $s_j$ is the undersized segment set, T is the minimum spanning tree;
Output: Collection of nodes of the oversized segment;
1.    Begin numbered list?
2.        If{e(u,v)|u ∈ $s_i$, v ∈ $s_j$}∩ $E^{MST} = \emptyset$ then// Edge e is the adjacent edge between oversized segment, $s_i$, and undersized segment, $s_j$, and endpoint u of edge e is a node that belongs to oversized segment and endpoint v of edge e is a node that belong to undersized segment,
3.        Δmin=∅;
4.      Else
5.        $\delta_{size} = \emptyset, \delta_{set} = \emptyset$ , n=0;//Initialization
```

```
6.        $E^{Psi} = \bigcup_{a,b \in s_i}\{e(a,b) | e(a,b) \in E^{MST}\}$
7.        **For all** $e \in E^{Psi}$ **do**
8.          $E^{Psi} = \{E^{Psi} \backslash e(u,z)\}$; //Cutting the edge e(u,z) where endpoint u
            of edge  e is a node that connected to undersized segment by the
            edge e(u,v) in line 2 and endpoint z of node e is the other endpoint
            of the edges belong to oversized segment.
9.          $\{\Delta u, \Delta z\} \leftarrow \mathbf{Disjoint - set}(\mathbf{E}^{Psi}, |\mathbf{T.V}|)$;// After cutting the edge
            u two segments $\Delta u$ and $\Delta z$ are obtained by the Disjoint-Set
            algorithm. One segment having node adjacent node, u, that is
            called $\Delta$u the other one is $\Delta$z that is not used.
10.         n=n+1;
11.         $\delta_{set}(n) = \Delta u$;
12.         $\delta_{size}(n) = |\Delta u|$;
13.         $E^{Psi} = \{E^{Psi} \cup e_{u \leftrightarrow z}\}$;//Restoring
14.       **End for**
15.       n*=$\arg\min_n \delta_{size}(n)$;
16.       $\Delta$min=$\delta_{set}(n*)$;
17.     **End if**
18.     **Return $\Delta$min**
19.   **End**
```

Algorithm 2.9 is utilized to calculate Δmin subset for the oversized segment in algorithm 2.8. The inputs of the Algorithm 2.9 are oversized segment, s_i , and undersized segment, s_j, and the minimum spanning tree, T. Line 2 checks the neighboring of the segments. If they are not neighbor, the Δmin set is empty. If the segments are neighbor, these segments are connected by a link, line 5 initializes the size of segment set, δ_{size}, and the segment set , δ_{set}, and the variable **n**. Line 6 returns the set $E^{Psi}$of links belonging to oversized segment, s_i. The link that contains node the adjacent node, u, and this node belongs to the oversized segment cuts from the E^{Psi} set (Line 8). After cutting, the oversized segment is divided into two segments. Disjoint-set function, Algorithm 2.5, returns these segments (Line 9). One of these sub segments contains adjacent node, u. This segment is qualified for segment modification. The sets δ_{size} and δ_{set} are updated (Lines 11-12). This loop is continued for all links of the E^{Psi} that contains adjacent node, u. Line 15 returns the minimum size of δ_{size}. Finally, in line 16 the corresponding node set Δmin is computed. Figure 12 illustrates the updated segment obtained by Algorithm 2.9.

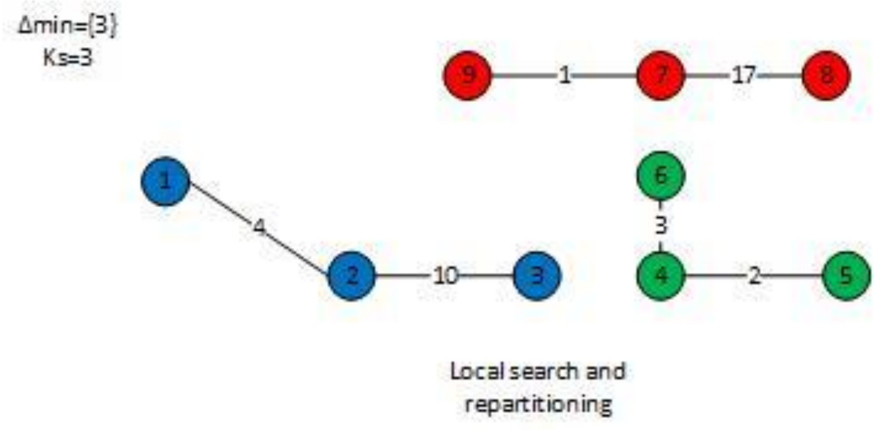


Figure 12: Updated segments of figure 11 obtained by algorithm 2.8 [1].

The node 3 in Figure 11 which is located in the oversized segment, {3,4,5,6} is shown by green color in Figure 11, is identified by the Algorithm 2.9. The inputs of Algorithm 2.9 are oversized segment, {3,4,5,6}, and undersized segment, {1,2}, and minimum spanning tree, T, the adjacent link is e (3,6) and the adjacent node is u=3. In line 6 the set of links belongs to oversized segment is initialized, $E^{ps_i} = \{(3,6),(6,4),(4,5)\}$. The link (3,6) is the only link that contains adjacent node, 3, from E^{ps_i} as a consequence, the for loop in Line 7 iterates only one time. In Line 8 the link (3,6) is cut from the E^{ps_i} set and in line 9 two segments are returns by the Disjoint-set algorithm, Algorithm 2.5, these set are {3}and {4,5,6}. The set that contains the adjacent node, {3}, is returned by the Find Δmin algorithm. This node is adjacent to the undersized segment, {1,2} shown by blue color in Figure 11. Algorithm 2.8 receives the Δmin set, {3}, from the Algorithm 2.9 and then checks the balancing condition in Line 10. In Line 11, node 3 is removed from oversized segment and added to undersized segment. The output of the Algorithm 2.8 is the updated set of segments, S= {{1,2,3},{4,5,6},{7,8,9}}.

2.3.3 Definition of Optimal TSP Problem

After the segments are initialized by the Algorithm 2.7 using LPP1, and repartitioned by the Algorithms 2.8, 2.9 solving ideal segmentation problem, the optimal trust nodes must be selected. The LPP2 is the optimal trust node computation problem that is described by (2.16)-(2.19) [1]:

$$\min_{X} \sum_{s\in S} \sum_{b\in B(s)} x_{sb} \tag{2.16}$$

$$\text{subject to}$$

$$\sum_{x_I \in X_I(l)} x_I \geq 1\,, \forall\, l \in L_{s\acute{s}};\ \forall s, s' \in S\,, \tag{2.17}$$

$$x_{sb} \in \{0,1\},\ \ \forall s \in S \text{ and } \forall b \in B(s), \tag{2.18}$$

$$\text{where}$$

$$X_I(l) = \{x_{sb}, x_{s'b'}\}, b \in B(s), b' \in B(s'), s \neq s', l = (b, b'), \tag{2.19}$$

where B(s) denotes the collection of bordering vertexes, $L_{s\acute{s}}$ denotes the set of inter-segment links between segment s and s', and $X_I(l)$ is the parameter set for bordering vertexes relevant to the inter-segment link, l.The output of the LPP2 is the quantity of trust nodes that the system needs to be protected against the attacks. The decision binary variable is $X = (x_{sb})_{\sum_{s\in S}|B(s)|\times 1}$, where b∈B(s),

$$x_{sb} = \begin{cases} 1, & \text{if } b\in B(s) \text{ is selected, } s\in S; \\ 0, & \text{otherwise} \end{cases} . \tag{2.20}$$

The objective function is given in (2.16) and inequality (2.17) ensures that all inter-segment links are covered by at least one trust system.

Algorithm 2.10: Pseudo code of trust node selection algorithm.

Optimal trust nodes placement algorithm

Input: S= {s1, s2, ….., sK}, G (V, E)

Output: V^{Trust};

```
1.  Begin
2.    For all s∈ S  do
3.      B(s)=∅;//Initialize bordering node sets
4.    End for
5.    For all s≠s' and s,s'∈ S do
6.      $L_{ss'} = \emptyset$;//Initialize inter-segment link sets
7.    End for
8.    For all e(u,v)∈ E do
9.      Find the segment x having node u;
10.     Find the segment y having node v;
11.     If  x≠y then
12.       $L_{xy} = \{L_{xy} \cup e\}$; // Updating inter-segment link set by adding link
          e(u,v);
```

```
13.         $B(x) = \{B(x) \cup u\}$; // adding node u to bordering node set x;
14.         $B(y) = \{B(y) \cup v\}$; // adding node v to bordering node set y;
15.       end if
16.     End for
17.     $V^{Trust} \leftarrow$
        Solve LPP2(inputs: the objective function and constraints (2.16) – (2.19), outputs: The vector X for binary decision variables )//This will select the trust node set
18.     Return $V^{Trust}$
19.   End
```

Algorithm 2.10 is used to identify the bordering nodes and select the trust nodes. The input of the algorithm 2.10 is a set of segments S which is the output of the algorithm 2.8 and the original graph. G (V, E). The output of the algorithm 2.10 is the selected trust nodes. Lines 2-7 are the initialization part of the algorithm. In this part the set of bordering nodes, B(s), per segment is initialized (Lines 2-4). For each pair of segments, the inter-segment link set, $L_{ss'}$, is initialized in Lines 5-7. For all links, e(u,v), in original graph, the segment, x, is identified that contains node u of the link e(u,v) and segment y is identified that contains node v of the link e(u,v) (Lines 9-10). If these segments are different, this link is known as an inter-segment link. Lines (12-14) update the bordering node set and inter-segment link sets. By using LPP2 the trust nodes are selected (Line 17). Figure 13 illustrates the intersegment links. These links are shown by the dotted line in Figure 13.

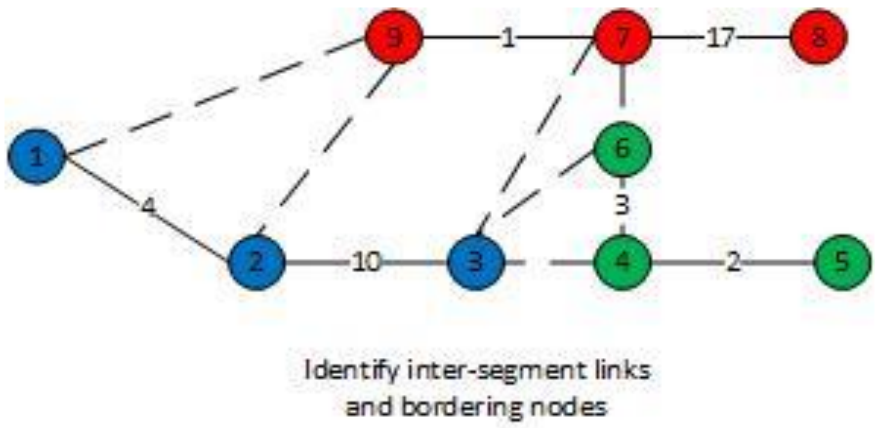


Figure 13: Identifying inter-segment links and bordering nodes [1].

Figure 13 illustrates the inter-segment links set, Algorithm 2.10 identifies the sets of the inter-segment links, Lss', and the set of bordering nodes, B(s), as follows:

$L_{s1s2} = \{(3,4), (3,6)\}$ and $L_{s1s3} = \{(1,9), (2,9), (3,7)\}$ and $L_{s2s3} = \{(6,7)\}$.

B(s1)={1,2,3}, B(s2)={4,6}, B(s3)={7,9}.

After initializing the bordering node sets and inter-segment links, the trust nodes, V^{Trust}, are obtained by the LPP2 in line 17. The inputs of the LPP2 are the objective function (2.16) and the constraints (2.16)- (2.19). The output is the vector X, $X = (x_{sb})_{\sum_{s \in S}|B(s)| \times 1}$, where b∈B(s). The selected trust nodes in Figure 14 are 3, 7 and 9.

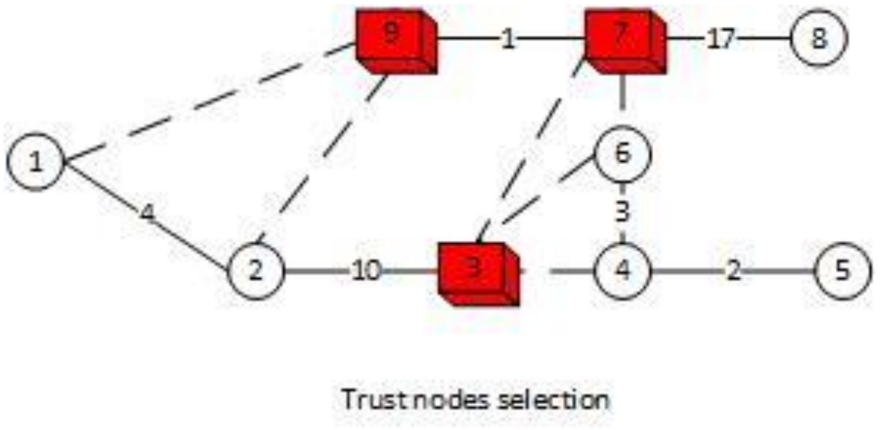


Figure 14: Finding trust nodes by algorithm 2.10[1].

Figure 15 depicts the flowchart of the optimal TSP problem solution algorithm. This flowchart has 5 blocks except starting and ending blocks. In block 1, SCADA network graph. G (V, E), and number of segments, K, are inputs of the Algorithm 2.7. In block 2, primary segments are computed by the Algorithm 2.7 (Segmentation problem); the output of the Algorithm 2.7 is the set of segments, S, and this is used as an input for the Algorithm 2.8 in block 3. Block 3 is about the local search process in Algorithm 2.8 to find the oversized, s_i, and undersized, s_j, segments. This block implements the ideal segmentation problem. Algorithm 2.9 is part of the Algorithm 2.8. The inputs of the Algorithm 2.9 are oversized, s_i, and undersized, s_j, segments and minimum spanning tree, T, ant the output is the set of sub-partition, Δmin, of the oversized segment. The functionality of the Algorithm 2.9 is to find the set of nodes from the oversized segments. The output of the block 3 is the updated set of segments. Algorithm 2.10 implements the trust node selection problem (Block 4). The inputs of the Algorithm 2.10 in block 4 are output of Algorithm 2.9, set of segments, and the input graph in block 1, G(V,E). Output of the block 5 is the selected trust nodes set (Block 5).

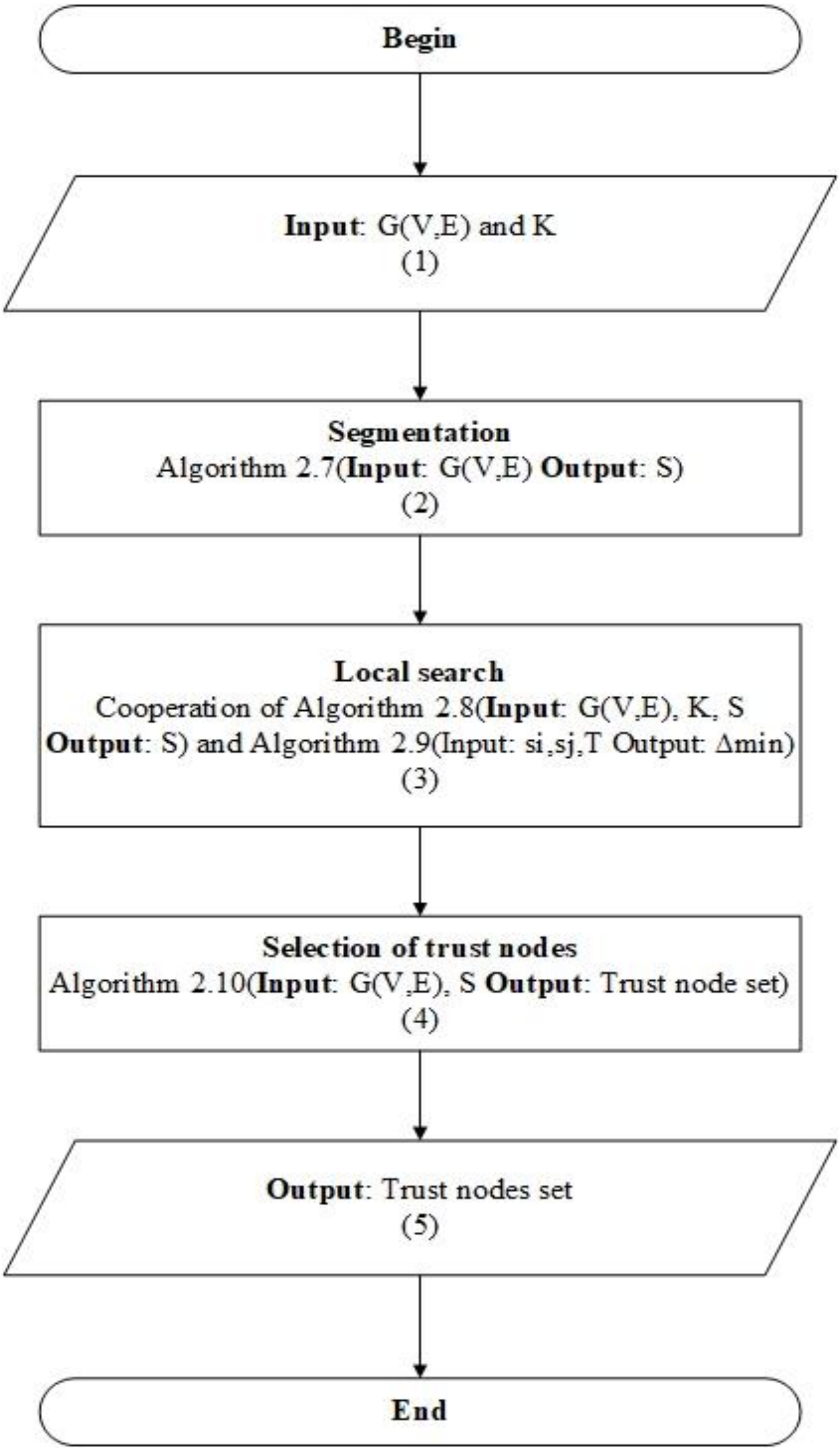
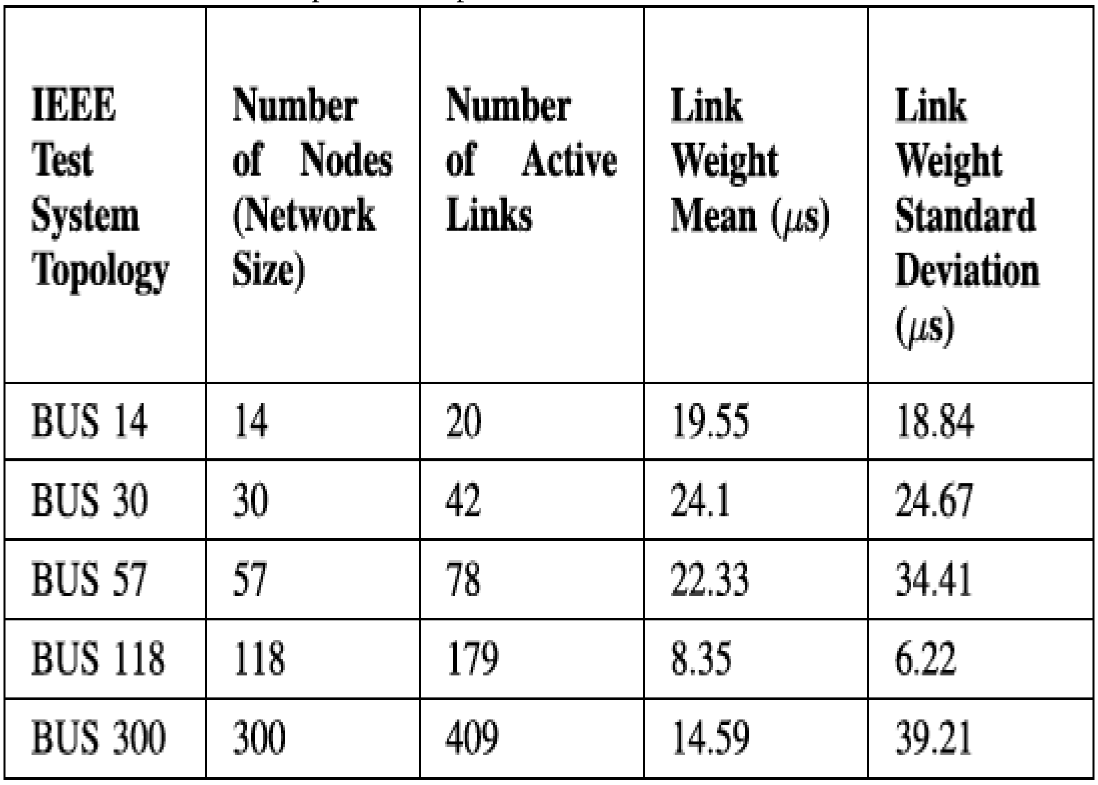

Begin
Input: G(V,E) and K
(1)
Segmentation
Algorithm 2.7(Input: G(V,E) Output: S)
(2)
Local search
Cooperation of Algorithm 2.8(Input: G(V,E), K, S Output: S) and Algorithm 2.9(Input: si,sj,T Output: Δmin)
(3)
Selection of trust nodes
Algorithm 2.10(Input: G(V,E), S Output: Trust node set)
(4)
Output: Trust nodes set
(5)
End

Figure 15: Flowchart of the algorithm for solving optimal TSP problem [1].

2.4 Experimental Outcomes

Case studies are conducted in [1] for the IEEE test system topologies [20]. The IEEE test cases represent the part of the American Electric Power Systems. Table 2 illustrates the overview of experimental parameters.

Table 2: Overview of experimental parameters [1]



IEEE Test System Topology	Number of Nodes (Network Size)	Number of Active Links	Link Weight Mean (μs)	Link Weight Standard Deviation (μs)
BUS 14	14	20	19.55	18.84
BUS 30	30	42	24.1	24.67
BUS 57	57	78	22.33	34.41
BUS 118	118	179	8.35	6.22
BUS 300	300	409	14.59	39.21

The topologies are divided into two parts on the base of the network size: large networks and small size networks. BUS 118 and BUS 300 are members of large networks. BUS 14, BUS 30, and BUS 57 are members of the small networks. Databases of the IEEE test system topologies include: bus number, load flow area, loss zone, circuit, branch resistance, branch reactance, base KV (Kilo Volt), load MVAR (Mega Volt Ampere Reactive), load MW (Mega Watt), minimum voltage and maximum voltage. Databases are stored as text files. The first seven fields of the first record of the IEEE test system topology of the BUS14 is shown in Table 3.

Table 3: Structure of the IEEE test system topology BUS14.

1	2	3	4	5	6	7
1st Node number of an edge Includes: 4 characters	2nd Node number of an edge Includes: 4 characters	Load flow area number includes: 1 character	Loss zone area number Includes: 1 character	number of parallel transmission Includes: 1 character	Transmission line Includes: 1 character	Branch resistance (Ω) Includes: 10 characters
1	2	1	1	1	0	0.01938

An electric power grid is an interconnected network to bring electricity from producers to consumers [21]. It contains generating stations, high voltage transmission lines and distribution lines.

Generating station, generator, converts the mechanical energy using steam turbines, gas turbines, water turbines into electrical power for use in an external circuit [22]. High voltage transmission line, electric power transmission system, moves the electrical energy from generating station to an electrical substation [23]. Electric power distribution is the final stage in the delivery of electric power, it carries electricity from the electrical substation to individual consumers [24].

In a smart grid environment, electric power grids are assumed to be accompanied by the representative SCADA communication networks. In a representative network, each link corresponds to a power grid branch and each node corresponds to a particular power grid bus. Links are weighted by the propagation delays. Figure 16 shows the IEEE BUS 14 test system topology with 14 nodes (buses) and 20 active links.

In Figure 16, diagram of IEEE BUS 14 is depicted. IEEE BUS 14 includes 14 buses which indicate the generator stations and 20 branches which indicate the transmission lines. The arrows in Figure 16 demonstrates the electrical loads, component or portion of a circuit that consumes electric power [25]. Each bus in a power system can be classified into three types [26]. First one is known as load bus. All buses in load bus having no generators. Second one is the generator bus. The buses that have generator are known as generator buses. The third one is the slack bus that it balances the active and reactive power in power system. IEEE BUS 14 includes 14 buses, correspond to node in graph, that buses 1,2,3 and 8 are generator buses and the other buses are load buses. The rectangle in Figure 16 demonstrates the transmission lines number, correspond to link in graph.

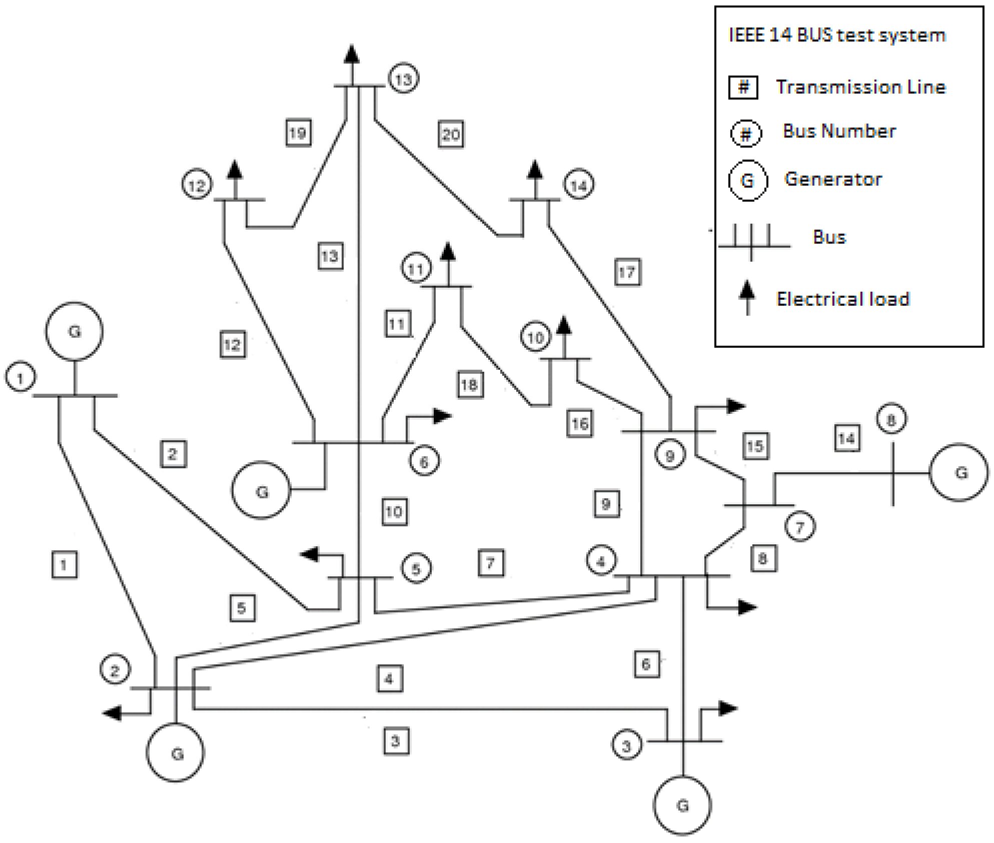


Figure 16 : IEEE BUS 14 test system.

The information that must be retrieved from the databases includes: node number in column 1 of Table 3, node number in column 2 of Table 3 and branch resistance in column 7 in Table 3. Brach resistance is used to calculate the propagation delay. Structure of the database record is as follows:

- First field defines the first node number of the link. This field includes 4 characters (including space characters). For example, in the first line of BUS 14 (Column 1 in Table 3), characters 1-3 are spaces and the fourth character is 1, as a consequence, the node number is equal to 1.
- Second field defines the second node number of the link. This field includes 4 characters (including space characters). For instance, characters 1-3 (Column 2 in Table 3) are spaces and the fourth character is equal to 2 as a result, the node number is 2. After these 2 steps, two nodes of the link, (1, 2), are found.
- Field 7 includes 10 characters (including space characters) and defines the branch resistance. In the given example the branch resistance is 0.01938.

Because propagation delays are not given in the text file, the following calculations are used to retrieve propagation delays for IEEE test system topologies:

1. Branch resistance (R) is retrieved from the previous steps.
2. Static resistivity (ρ) is a measure of how strongly a material opposes the flow of electric current. Aluminum wire with an iron core is assumed in [3], which has a resistivity value of 2.50188E-8 Ωm.
3. *Area* is the cross-sectional area of the material in square meters (m^2). In [3], it is selected 0.00080642 m^2 as a typical value.
4. The line length of a piece of material is measured in meters. The line length is obtained by equation (2.21).
5. In [3], fiber optics cables are used in the communication line, therefore, the speed of light, 299792458 m/s, in fiber optics cable, fiber optic operates 99.7% speed of

light, is used in equation (2.22) to obtain the time or latency from the line length (2.21) [3].

$$\text{Line\_Length} = \frac{\text{R}\times\text{Area}}{\rho} \quad \text{m (meters)}, \tag{2.21}$$

$$\text{Time} = \frac{\text{Line\_Length}\times 3}{\text{speed of light}} \quad \text{s (seconds)}, \tag{2.22}$$

For instance, the propagation delay between nodes 1 and 2 of IEEE BUS 14 power system is calculated as follows:

Example 3: Computation of propagation delay between node 1 and node 2 of IEEE BUS 14.

R=0.01938 Ω. // retrieved from field 7 of Table 3.

Area=0.00080642 m^2.

$\rho = 2.50188\text{E} - 8\ \Omega\text{m}$.

Line_Length=$\frac{0.01938\times 0.00080642}{2.50188\text{E}-8} = 624.667034$.

Time=$\frac{624.667034\times 3}{299792458} = 6.250\text{E} - 6\ \text{s (seconds)} = 6\ \mu\text{s (micro seconds)}$.

The rest propagation delays are obtained in same way of Example 3 which are specified in Figure 17.

Figure 17 shows a graph of IEEE BUS 14 test system topology. In the graph, nodes correspond to the buses, generators or electrical loads, links represent communication between buses, transmission lines, and link's weights represent propagation delay in micro seconds.

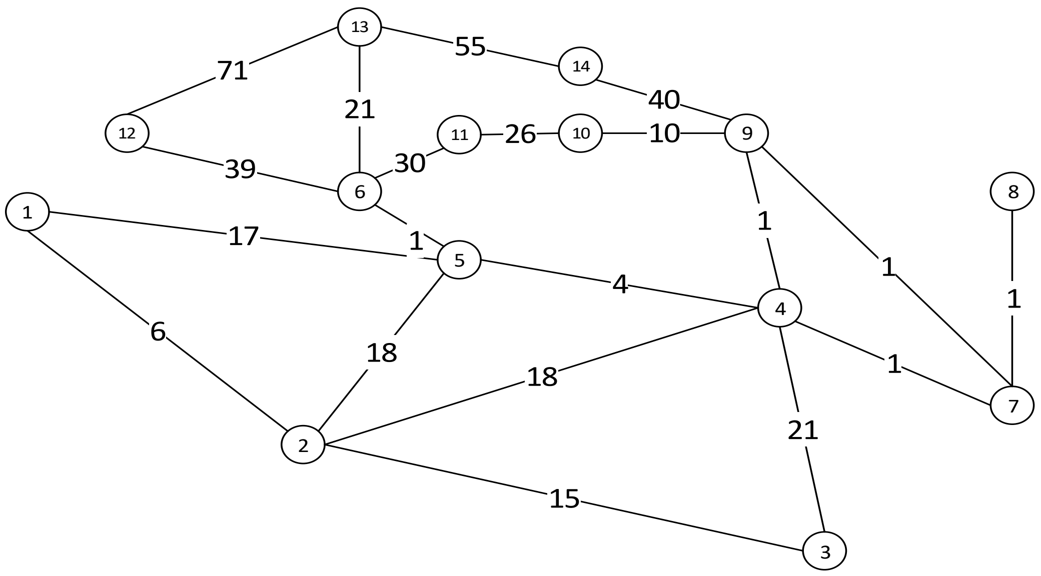


Figure 17: Graph of IEEE BUS 14 test system topology presented in figure 16.

The designed method implementation employed MATLAB optimization toolbox. The IEEE test system models are utilized as SCADA network graphs. Small networks are divided into 3 to 6 segments with increment of 1. Large networks are split into 5 to 30 segments with increments of 5. The mean value of the segment size differs between 2.33 and 19 in small size networks and also, the range of mean value of the segment size is between 3.93 and 60 in large networks. In [1], all the examinations are implemented on a PC system equipped by RAM 4 GB and Intel core i3, 3.30 GHz processor.

In [1] coefficient of variation, also known as relative standard deviation, RSD, of segment size [27] is chosen as a metric. Equation (2.23) demonstrates the coefficient of variation, CV, formula [27].

$$CV = \frac{\sigma}{\mu}, \tag{2.23}$$

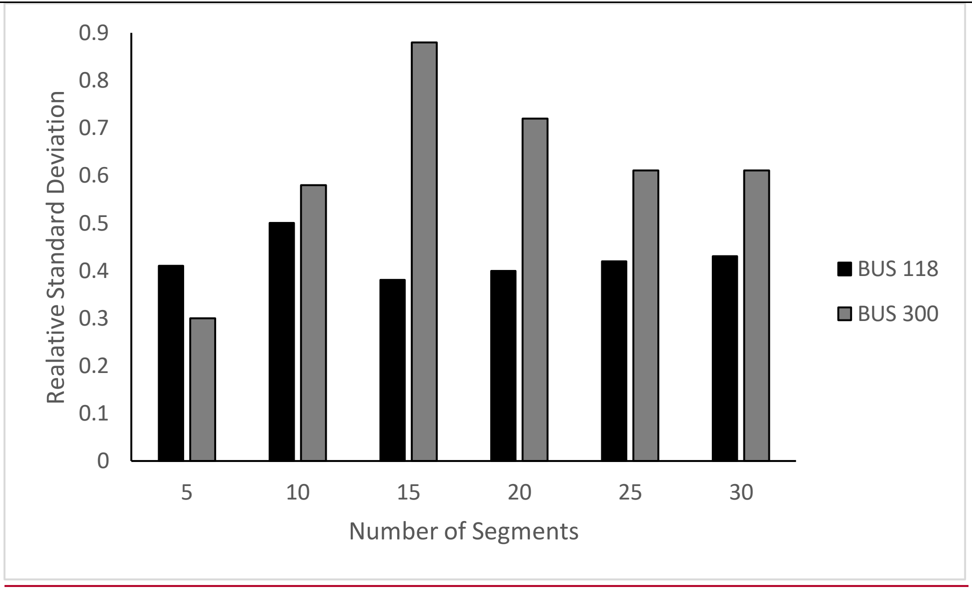
where σ is the standard deviation and μ is the average. Equation (2.24) illustrates the standard deviation formula [28] for values x[1..N]:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N-1}} \quad , \qquad (2\ .24)$$

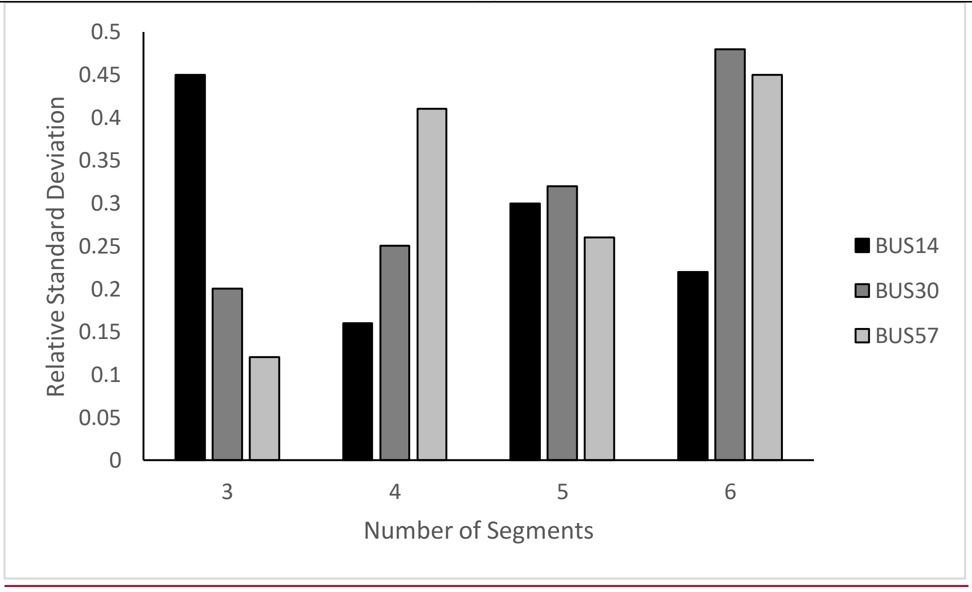
. Equation (2.25) defines the mean of x[1..N] values :

$$\mu = \frac{1}{N}\left(\sum_{i=1}^{N} x_i\right) \ , \qquad (2.25)$$

Figure 18 demonstrates the relative standard deviation of the calculated segment sizes for small (Figure 18(a)) and large (Figure 18(b)) networks.



(a)

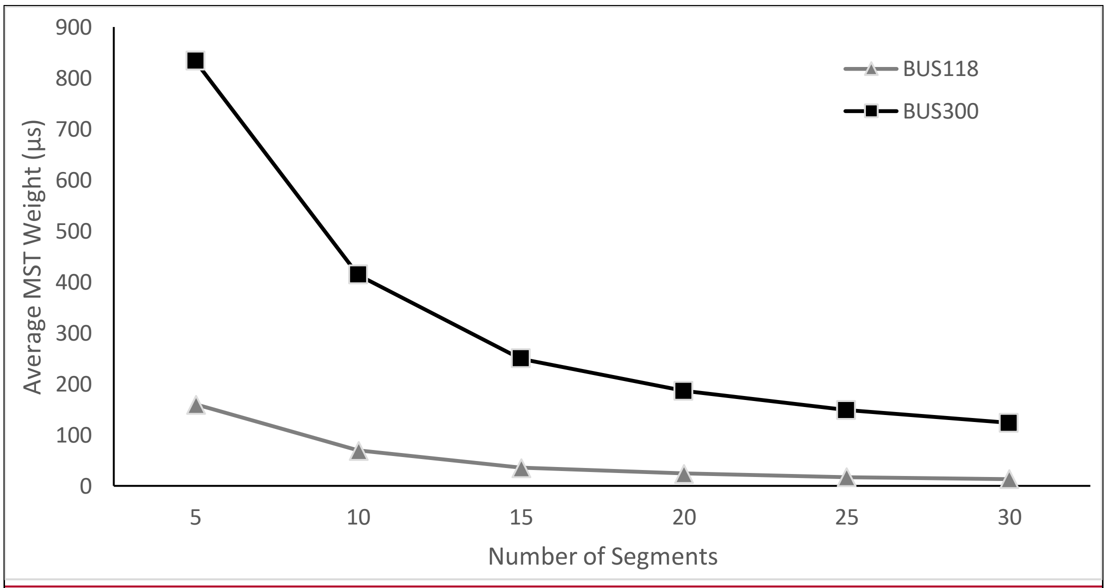


(b)

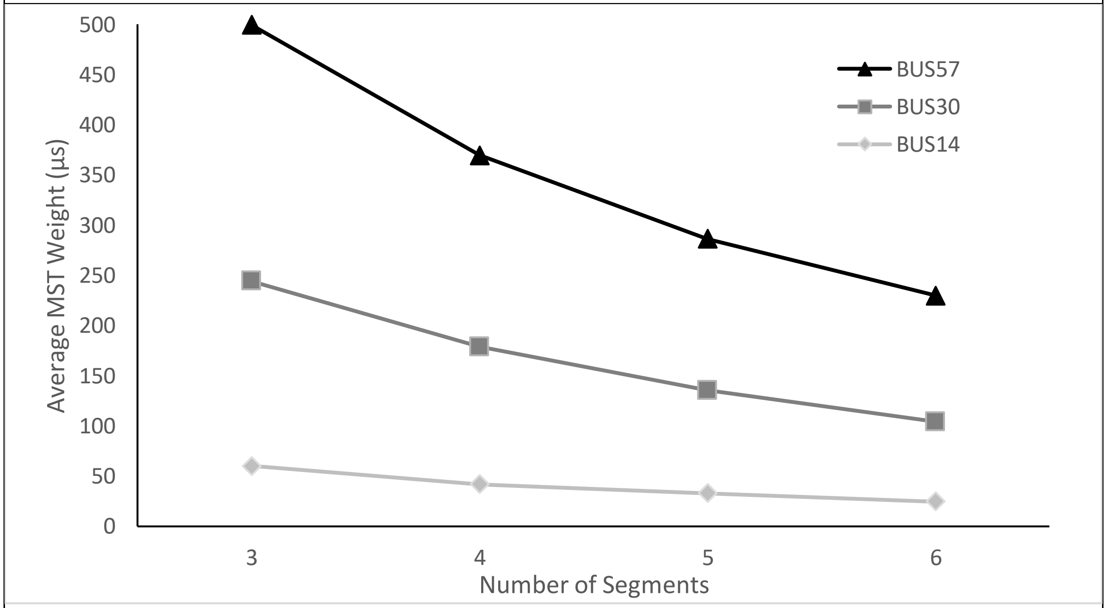
Figure 18: Relative standard deviation of the calculated segment sizes. (a) Large networks. (b) Small networks [1].

Relative standard deviation of the computed segment sizes is less than 0.5 for small networks and is less than 1 for large networks. As the Figure 18 shows, it is obvious that the coefficient for large network is higher than for small network and it is because of network size. In large networks segments have lower size balance and as a consequence, the relative standard deviation of the calculated segment sizes is larger.

In Figure 19, the average MST weight (2.25) is the metric for the geographic dispersion. Figure 19(a) is for small networks and Figure 19(b) is for large networks. In both of them, it is clear that by increasing the number of segments the average MST weight is reduced. The segment sizes and quantity of segments are related, and it means that increasing number of segments causes decrementing of the segment sizes. As a consequence, there is a decrement in the average of MST weight.



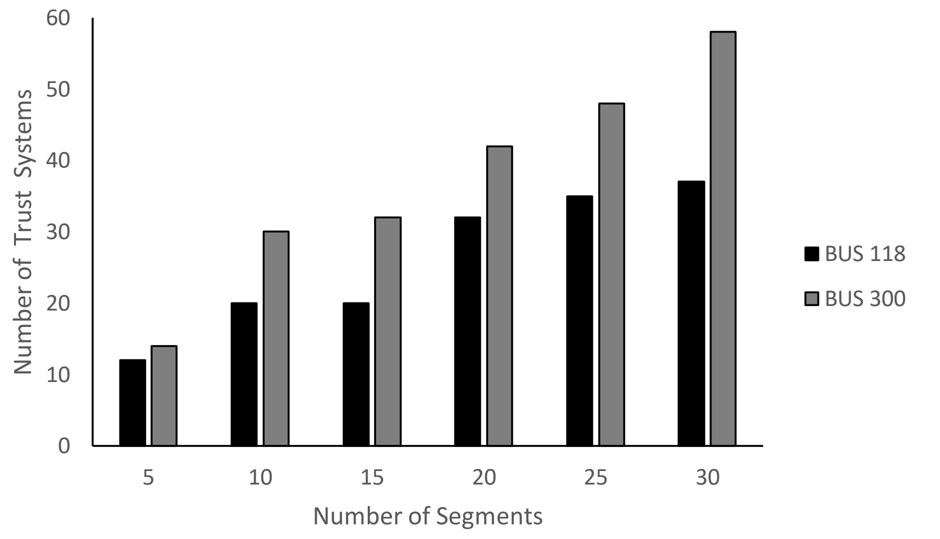
(a)



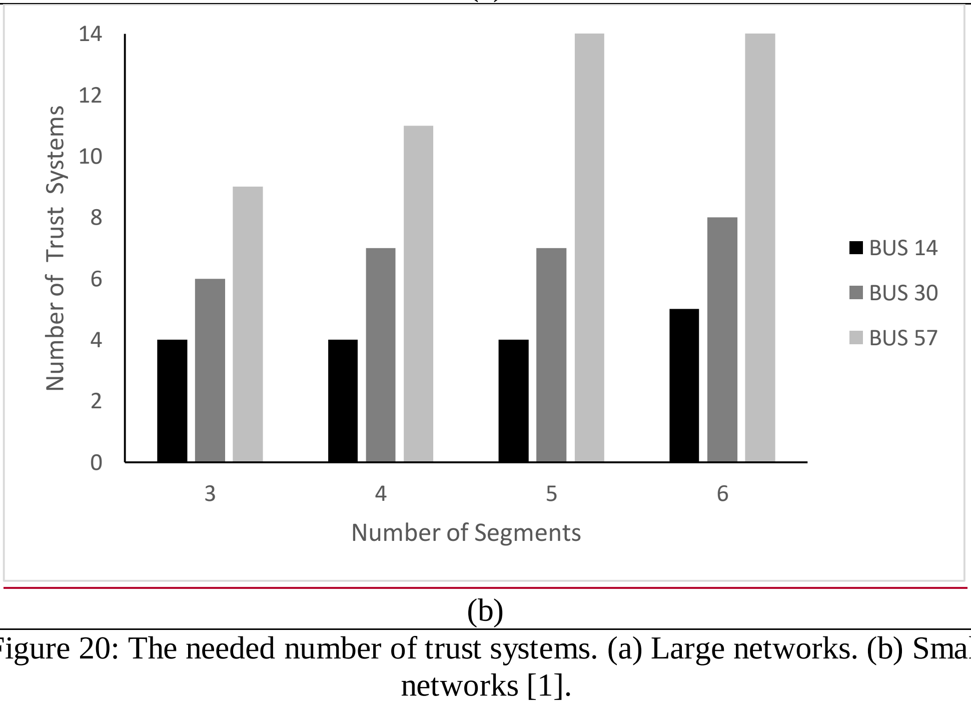
(b)

Figure 19: The average MST weight of the calculated segments. (a) Large networks, (b) Small networks [1].

Figures 20 shows the desired quantity of trust systems, calculated by the Algorithm 2.10, for the proposed scheme. In both cases, the bar chart shows that required amount of trust systems rises with the amount of segment increasing. The amount of increments is higher for larger networks.



(a)



(b)

Figure 20: The needed number of trust systems. (a) Large networks. (b) Small networks [1].

2.5 Problem Definition

SCADA networks are used to monitor and control the input and output traffic to protect the smart grid networks against the intrusions, malicious activities and other bad activities that harm the smart grid networks. For this reason, trust systems are deployed by the smart grid operators to monitor traffic packets. Trust systems consist of firewalls and intrusion detection systems (IDS).

The objective of optimal TSP problem is to optimize the cyber-security of smart grid networks. As trust systems contain specialized software and hardware agents, it is expensive to deploy them through entire of network and also trust systems cause delay through the network. In [1], optimal TSP problem is proposed to optimize the security by minimizing the operational expenditure and capital expenditure. As a consequence, minimum number of trust nodes is selected and equipped by the trust systems. These nodes are known as trust nodes. Optimal TSP problem provides optimize security and minimize the cost.

Methods that are used to solve the optimal TSP problem comprise segmentation algorithm, local search algorithm and trust node selection algorithm. Segmentation problem is the main part of the optimal TSP problem. As smart grid networks, power grid systems, are geographically distributed the network is divided into small networks to restrict the spreading of cyber-attacks. Segmentation is based on the MST it means that in each segment the distance between components is minimal. Local search is used to uniform the size of the segments in number of nodes.

Only bordering nodes can host the trust systems. As a consequence, due to the budgetary minimum number of trust nodes are selected by the trust node selection algorithm. The constraint of trust node selection algorithm is that all inter-segment links must be connected to at least one trust node. As a result, if in one segment a bad activity, malicious traffic and other types of attack is took place then it can not distribute to other segment because segments are connected through the inter-segment links which they are equipped by at list one trust system.

We used IEEE test system topologies, categorized into small and large networks, as database to analyze the performance of optimal TSP problem. We use coefficient variation to measure the segments size. The coefficient of variation for small network is smaller than coefficient of variation for large networks. It means that segments in small networks are more balanced in number of nodes. By increasing the number of segments, the required number of trust systems increases, and the average of MST weighs decreases.

Chapter 3

DESIGN, IMPLEMENTATION, AND TESTING OF OPTIMAL TSP SCHEME

In this chapter, we explain design and implementation of the codes. This chapter includes three sections. In Section 3.1, design and implementation of proposed optimal TSP scheme [1] are discussed. In Section 3.2, testing of the developed optimal TSP scheme is discussed.

3.1 Design and Implementation of Proposed TSP System in [1]

As mentioned in Chapter 2, the optimal TSP scheme is based on three algorithms (Figure 14). Inputs of this software are divided into two parts, the first one is manual input and the second one is the text files, based on IEEE test system topology [20]. The source codes and the databases are shown in the Appendix A. After explanation of the codes, we will discuss the improvement part of the optimal Trust System Placement (TSP) scheme in smart grid SCADA networks program due to the minimization of the dispersion of the trust system number over the network segments that improves uniformity of the trust systems placement. Utilities that we used, are Microsoft Visual Studio.Net Enterprise 2015 and Matlab R2016a. This application is implemented by Microsoft C#.net which is an elegant and type-safe object-oriented language. Matlab is used to solve the LPP1 and LPP2 problems. All the examinations are implemented on a PC system equipped by an Intel core i7 2.10 GHz CPU and 8GB RAM and Microsoft Windows 10 64-bit operating system.

3.1.1 Design of Optimal TSP Scheme [1]

Figure 21 demonstrates 8 blocks of process to implement the proposed optimal TSP in smart grid SCADA networks. These blocks are shown as follows:

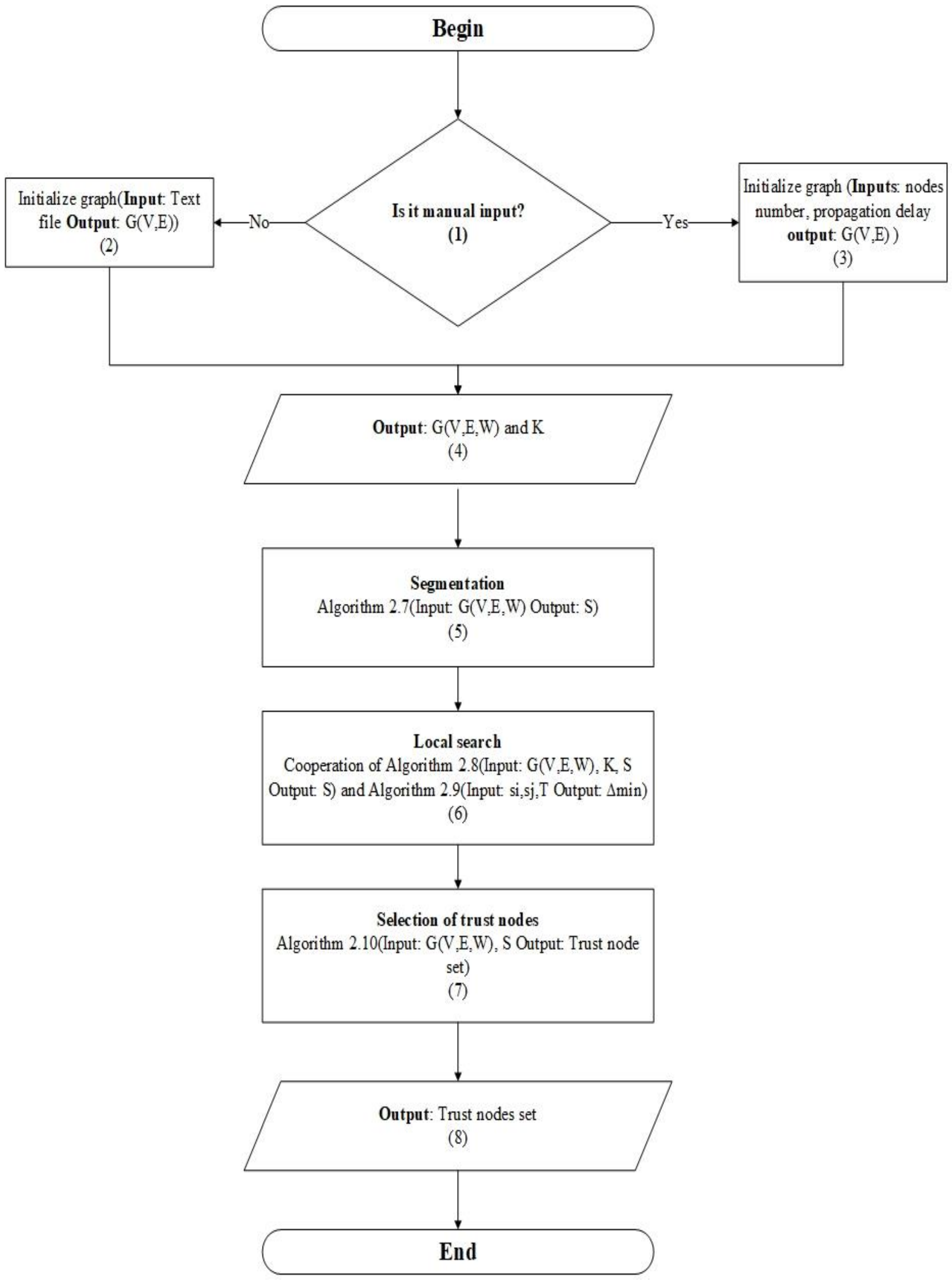

Begin
Is it manual input? (1)
No
Yes
Initialize graph(Input: Text file Output: G(V,E)) (2)
Initialize graph (Inputs: nodes number, propagation delay output: G(V,E)) (3)
Output: G(V,E,W) and K (4)
Segmentation Algorithm 2.7(Input: G(V,E,W) Output: S) (5)
Local search Cooperation of Algorithm 2.8(Input: G(V,E,W), K, S Output: S) and Algorithm 2.9(Input: si,sj,T Output: Δmin) (6)
Selection of trust nodes Algorithm 2.10(Input: G(V,E,W), S Output: Trust node set) (7)
Output: Trust nodes set (8)
End

Figure 21: Block diagram of process to implement the proposed TSP in smart grid SCADA networks.

Block 1 is a decision process to select the graph. Initialization of the graph is divided into 2 types, initializing graph of IEEE test system topologies via text file (Block 2) and manually initialize the graph (Block 3). The output of the Blocks 2 or 3 is used as an input (Block 4) for the segmentation problem (Figure 15 Block 2). Block 5, implements the code to execute the Algorithm 2.7 and the output of block 5 is used as an input of block 6. In block6, the Algorithms 2.8 and 2.9 are implemented to run the local search procedure. The output of the block 6 is the updated segment sets and used as an input for the selection of trust nodes procedure (Block 7). Block 7 implements the Algorithm 2.10 and the output is the set of trust nodes that must host the trust systems. The source codes of theses blocks are described in Section 3.1.2.

3.1.2 Implementation of optimal TSP scheme

Appendix A demonstrates the source codes of the optimal TSP program. This program includes Edge, Graph, Create_Graph_text, Create_graph Minimum_Spanning_Tree, LPP, Delta_min, Bordering_node_details, button1_click, and Trust_Node_selection classes. Each process in Figure 21 includes several classes.

1. The process of initializing graph in blocks 2 and 3 implement the input process uses classes: Edge, Graph, Create_Graph.cs, Creat_Graph_text and button1_click event.
2. Segmentation process (Block 5) uses classes: Minimum_Spanning_Tree.cs and LPP.cs.
3. Local search process (Block 7) uses classes: Delta_min and Form1.cs .
4. Selection of trust nodes process (Block 9) uses classes: Bordering_node_details.cs and Trust_Node_selection.cs.

The first part of the implementation is to define the attributes of the graph which include: edge, source/destination node, weight (propagation delay). These attributes are obtained by the process in block 2 or 3. Block 1 decides which block 2 or 3 might be run. Figure 22 shows the main form appearance.

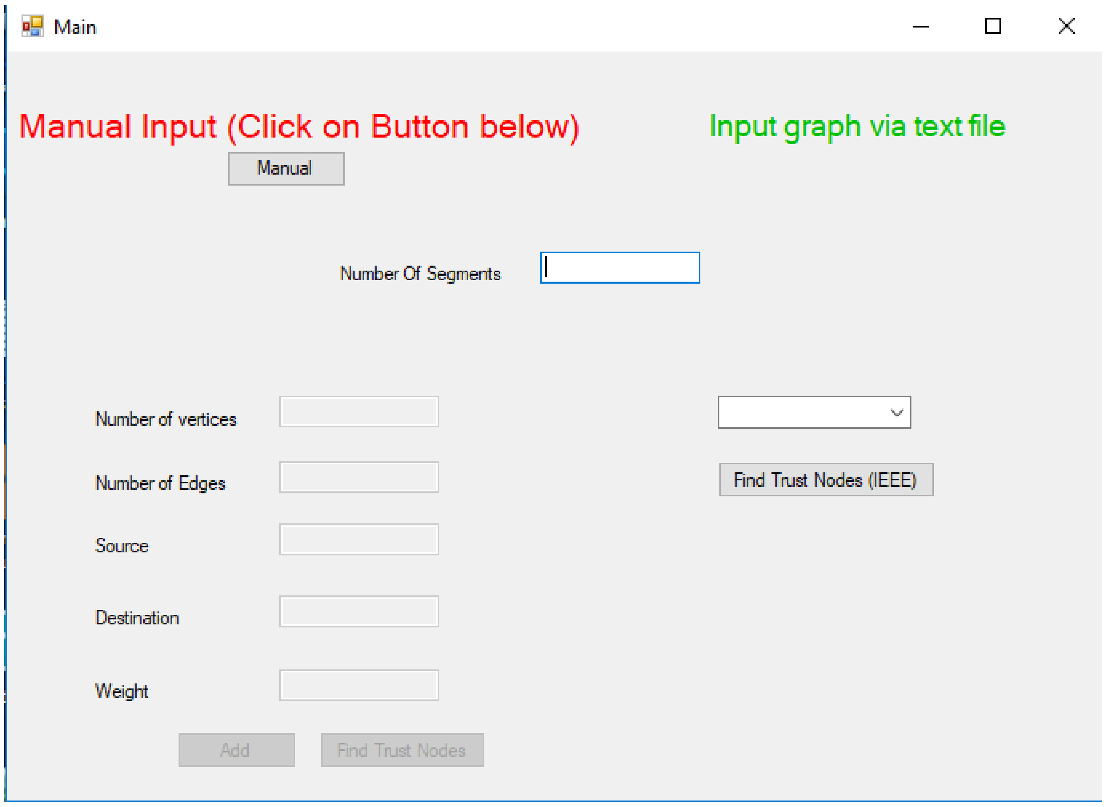


Figure 22: Main form of the program.

Figure 22 illustrates that the default input type is by the text file because the elements of manual input are disabled. By clicking on the combo box and choosing the databases (BUS14, BUS 30, BUS57, BUS 118 and BUS300), the decision process in block 1, decides to execute the process of block 2 in Figure 21. Otherwise, if the “Manual” button is clicked then the process of block 3 is executed (Manually input). The manual input includes number of vertices (nodes), number of edges (links), source (node number), destination (node number) and weight of the link. The graph is undirected, the source and destination value just denote the two endpoints of the link. Classes Edge, Graph and Create_Graph are used in the both processes. Block 2, except the mentioned classes, executes the Create_Graph_text and block 3 executes the button1_click event. Figure 23 illustrates the flowchart of the input graph procedure with the use of a text file. Figure 24 shows the main form for the manual input graph, and Figure 25 depicts the flowchart of the procedure of manual graph initialization.

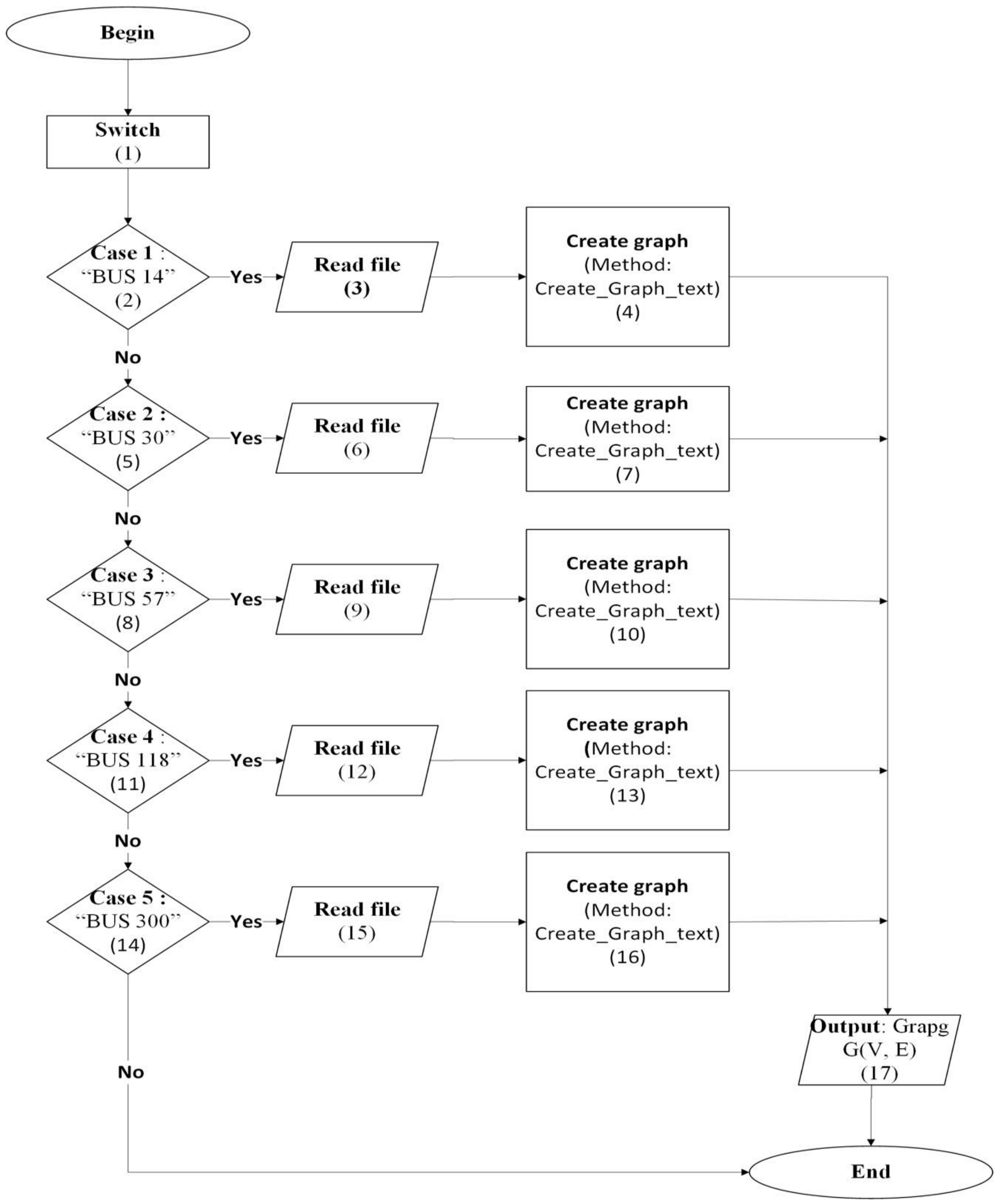

Begin
Switch (1)
Case 1 : "BUS 14" (2)
Yes
Read file (3)
Create graph (Method: Create_Graph_text) (4)
No
Case 2 : "BUS 30" (5)
Read file (6)
Create graph (Method: Create_Graph_text) (7)
Case 3 : "BUS 57" (8)
Read file (9)
Create graph (Method: Create_Graph_text) (10)
Case 4 : "BUS 118" (11)
Read file (12)
Create graph (Method: Create_Graph_text) (13)
Case 5 : "BUS 300" (14)
Read file (15)
Create graph (Method: Create_Graph_text) (16)
Output: Grapg G(V, E) (17)
End

Figure 23: Flowchart of the algorithm providing input of graph via text file.

Block 1 is a switch statement that chooses a single switch section to execute from a list of candidates based on a pattern match with the match expression [29]. It has 5 switch selections for IEEE test system topologies. By choosing one of the IEEE test system topologies from the combo-box in Figure 22, one switch selection is executed (Blocks 2, 5, 8, 11 and 14) and all lines of the text file will be read by the "File.ReadAllLines()" method and stored in an array of string [30]. In the next block, the array of string is parsed by the create_graph_text method to extract the attributes of the graph in text file (Table 3). Finally, the output is the SCADA network graph object. By clicking on the button (manual) the manual input graph is activated (Figure 24).

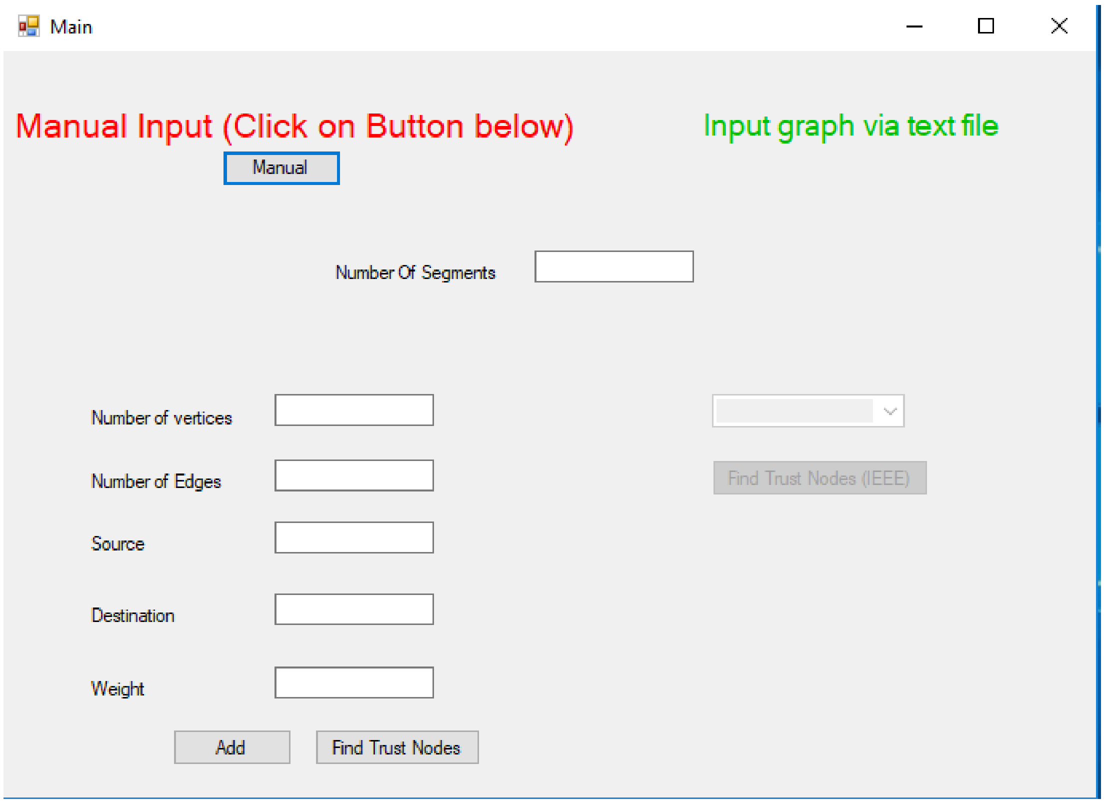


Figure 24: Manual graph input screen.

Figure 25 depicts a flowchart of the procedure of manual input of the graph. In block 1, the number of vertices (nodes) and number of edges (links) are read from the textboxes. The array of the object edges is created in block 2. In block 3, the variable **i** is set to zero. This variable is used to count the number of edges (links). The value of the weight, source (node number 1) and destination (node number 2) of the edge is read and if the value of the variable **i** is smaller than the number of edges (Block 5) then these values are stored to the attribute of the graph (Block 6). After adding these values, the value of **i** increases by 1. If the condition in block 5 returns false (i>number of edges), the graph is created by the Create_graph method (Block 8). The output is an undirected graph. Below, the classes which implement the input graph are discussed.

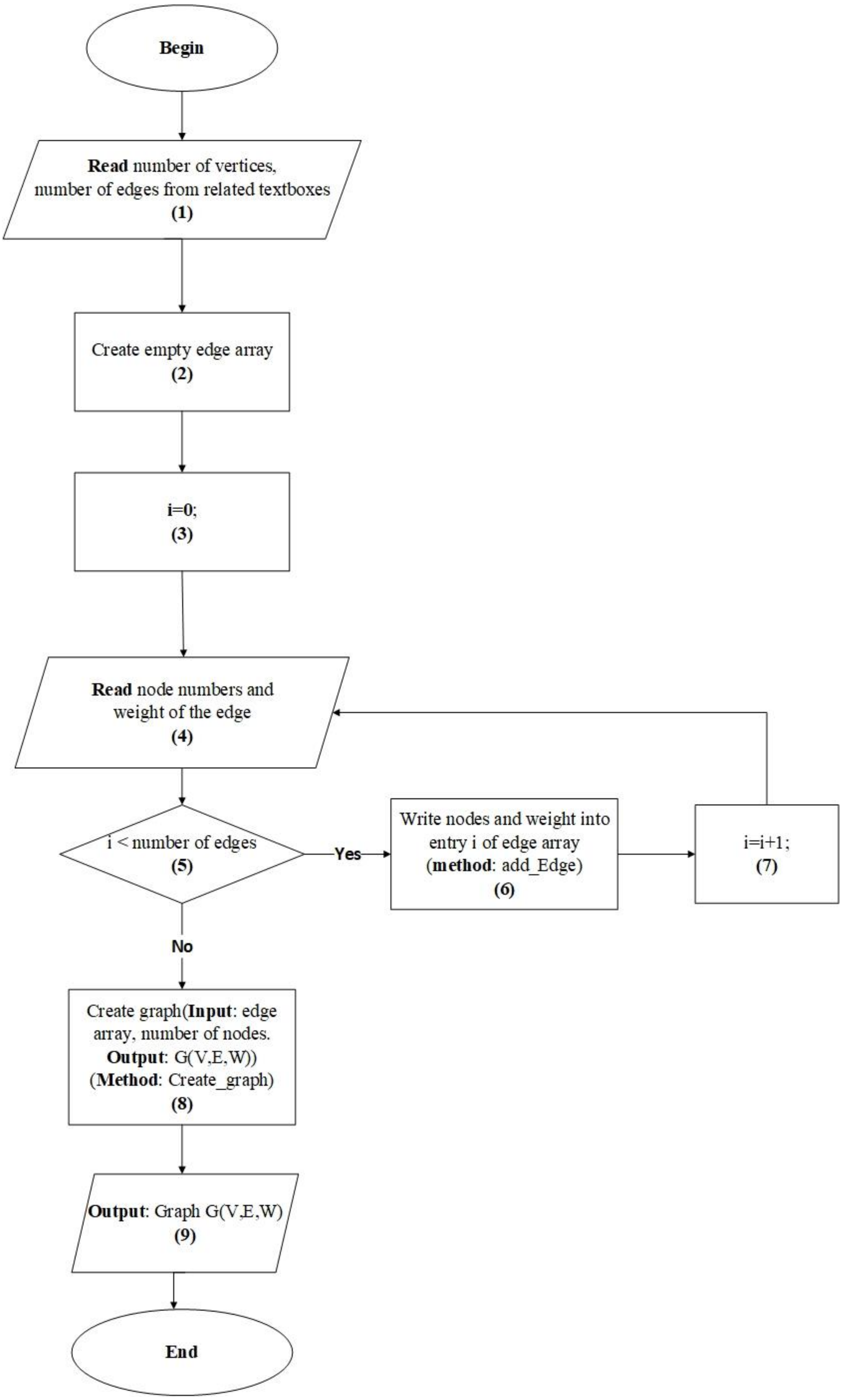

Begin
Read number of vertices, number of edges from related textboxes (1)
Create empty edge array (2)
i=0; (3)
Read node numbers and weight of the edge (4)
i < number of edges (5)
Yes
Write nodes and weight into entry i of edge array (method: add_Edge) (6)
i=i+1; (7)
No
Create graph(Input: edge array, number of nodes. Output: G(V,E,W)) (Method: Create_graph) (8)
Output: Graph G(V,E,W) (9)
End

Figure 25: Flowchart of the algorithm allowing of manual graph creation.

Defining class of Edge: Appendix A1 shows the class Edge.cs. It has 8 attributes and includes: Source, Destination, Weight, minDegree, maxdeg, normalizeWeight, we, Active, delta_Seg and Intersegment lines (3-12). There is one method, add_Edge, which initializes the attributes Source, Destination, Weight, and Active, which is a Boolean variable. The value of this variable at the beginning is equal to true. This method is used to add the value of the destination, source and weight to the created object of edge.

Defining class of Graph: Appendix A2 is about a class Graph. Lines 3-8 define the attributes which are related to the number of vertices, number of links, Edge properties and two constraint variables that are used to calculate the weight of the edge (propagation delay) [3] and they are defined in Section 2, (2.21)- (2.22). It includes seven methods. Lines (9-13) initialize the graph attributes such as: vericesCount, number of nodes, and edgesCount, number of links, and the array of links, ed, for manual inputs. Lines 14-28 is a method to compute the mean value of the links (2.25). Lines 29-50 defines a method, standard_Deviation, to computes the standard deviation of segments in size. Inputs are segments and the mean weight and the output is the standard deviation of the computed segment size (2.24). Line 51-56 define a method, coefficient_Variation, to computes the coefficient of variation of the segment size or of the trust nodes over the segments (2.23). Inputs are standard deviation and mean value. Lines 58-77 is a method to find the remote nodes, disconnected power grids in IEEE Buss300 are connected by the remote nodes and remote nodes share and exchange the power excess. This method is used only for BUS300. Appendix B6 show the information of remote nodes data. Table 4 illustrates the first row of the remote node file. I just need to retrieve a data from field 1, remote node number, and from field 19, node number. If their values are different the values are swapped.

Table 4: First row of the remote node text file for BUS300.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	0	1.0284	5.95	90	49	0	0	115	0	0	0	0	0	0	1

Defining the button1_Click event (Adding link's information for manual input): This method is run when the button by the name Add is clicked. Appendix A3 shows the codes of the button1_Click which executes the manual graph input procedure (Figure 25). Lines 5-10 initialize an empty array of edges. By clicking the Add button, Line 12 checks that this edge information does not exceed the number of edges (Figure 25, Block 5). Line 14 adds the edge attributes to the array in element with index number i. If variable i exceeds the number of edges, then the method create_Graph returns the output graph (Line 28).

Defining the function of Create_Graph_text (Input graph by using text files): This method executes in the class of the Graph. Appendix A4 shows the functionality of the creat_Graph_text method. The inputs of this method include: array of the lines of the text file, number of edges and number of vertices. Lines 3-19 are the initializing. In line 8 the remotenode function executes if the input text file is IEEE BUS300. Lines 20-75 is a loop that read characters of lines one by one from the input text file (database). In each line, characters 1-4 indicate the first node of the link (Line 28). In Line 30, there is a condition that checks the ASCII code of **space** (ASCII code of space=32): if it is not equal to ASCII code of the **space** then it is added to the **source** variable. Line 33 reads characters 6-9,and sets the value of the **dest** variable. The characters in range of the positions, [20-29], of the line indicate the branch “resistance”. Lines 45-61 is executed if the selected text file is BUS300. If the remotenode and node number are different they will be swapped. Line 41 sets the value of the **weigth,** resistance**,** variable. Lines (62-72) calculate the propagation delay of the link (2.21) - (2.22).

The segmentation process (Block 5) includes 2 classes as follows:

Defining class of minimum_spanning_tree: Appendix A5 defines MST. In this class, I used Kruskal algorithm, Algorithm 2.6, to compute the minimum spanning tree, Disjoint-set algorithm, Algorithm 2.5, to extract the segments and one function to compute the minimum degree of the links. Line 4-9 implements the Find algorithm, Algorithm 2.3. in line 6 if the node does not point to itself then the Find method calls itself to find the parent of that node, go up the tree to find the parent. Lines 10-29 implements the Bubblesort algorithm, Algorithm 2.1. The Exchange method in line 10 is used to swap the elements in the array of the link. Lines 30-43 implement the Union function, Algorithm 2.11. Lines 56-91 is the Disjoint-set algorithm, Algorithm 2.5. The input is the object graph G and the output is the segments. Lines 92-133 is Mst_Graph method that implements the MST-Kruskal algorithm, Algorithm 2.6, and the input is the object of the input graph, G(V,E), and the output is the minimum spanning tree, Mst.

Defining class of LPP: This class (Appendix A6) is used to solve the integer linear programming LPP in [1]. It has two functions LPP1 and LPP2. Line 4 is used to make a connection between Microsoft Visual Studio and Matlab. Lines 5 and 6 define two constraints (α=1, β=0.5) which are discussed in Chapter 2. In Matlab, all the variables are double, and also, they are defined as matrices. Hence, I defined a 2-dimensional arrays, matrices, and the attributes of the indexes of the matrix are double, Lines 9-16 are arrays of LPP2 and Lines 66-73 are arrays of LPP1. Line 7 is a method to solve the LPP2. The inputs are number of bordering nodes, number of inter-segment links and the set of inter-segment link and the set of bordering nodes. Lines 9-12 initialize objective function array (2.16), f, and the equation constraint array, intcon, and inequality constraints array (2.17), A, b. Lines 13-16 initialize the arrays to send to the Matlab application. Lines 17-36 the value of the arrays are set. Line 31-32 call the method Find_for_Array_A, Lines 45-62, to find the segment that having endpoints of inter-segment link. The object of array res is initialized. This object stores the returns answer from the Matlab application (Line 38). In line 40 the arrays are sent to the Matlab by the method Feval. The first input of this method set to "intlinprog" it means that the answer is a binary vector, {0,1}. In line 42, first index of the array **res** is the returned answer by the Matlab application and store in the array sln. Line 43 is the output of the LPP2 method. Lines 63-130 implement the LPP1. The structure of the code is same as the LPP2.

Implementing the **local search** process (Block 6) needs to implement the **while** loop in Algorithm 2.8. This loop finds the oversized segment and undersized segment. If these segments are adjacent, then Line 10 calls the delta_computation (Algorithm 2.9) method from the delta_min class to find the set of the nodes. This set will be removed from the oversized segment and added to the undersized segment. Below I am going to describe the functionality of the local search algorithm, button2_click event in form1 class, and FindΔmin procedure, delta_min class.

Defining class of Delta_min: Appendix A7 illustrates the source codes of the class Delta_min. This class defines Δmin computation algorithm (Algorithm 2.9) in chapter 2. Inputs of function (delta_computation) are oversized, S_i, and undersized segments, S_j, and minimum spanning tree, Mst, and minimum spanning tree of the segments, Mstss, these 4 arguments are shown in Line 3. Lines 4-17 initialize the variables such as: a set for delta_min, line 6, an adjacent node u, line 6, an array for Epsi, Line 9, an array for the links that are cut, line 10, an object f minimum_spanning_tree class to use the disjoint-set method, line11, segment set for adjacent node, line 13. Lines 18-32 check the adjacency of the segments. There is no adjacency if the value of the variable (u=999) does not change; otherwise, lines 42-76 extract the links set belonging to the oversized segments (Lines 42-76). All links that belong to node u are removed and collection of segment (Segment_set_U) will be updated (Lines 81-107). All the elements of the collection set of the segment (Segment_set_U) will be checked if the nodes of the segment include node u then, that segment is the output of the delta_min(Lines 108-120).

Defining the button2_click event: Appendix A8 demonstrates the Lines 1-10 in Algorithm 2.7 and the lines 1-20 in Algorithm 2.8 in Chapter 2. In line 3 the value of the variable phi is set to 1, this variable is used in while loop, line 28. In lines 5-8 the number of segments, K, and the object of the segment are initialized. In line 11, kruskal method (Appendix A5) is executed to obtain the minimum spanning tree, MST. Minimum degree of the links is computed in line 13. The link weight standard deviation and link weight mean are computed in lines 16-17. Line 21 executes the LPP1 and store the output data of LPP1 to the ESS, links of segment set. In line 26, the segments are returned by the disjoint-set algorithm, Segmentation method executes the disjoint set method. The variable Ks, limitation variable in Algorithm 2.8, is set in line 27. Lines 28-66 of Appendix A8 implement the local search to find the oversized segment and undersized segment. Line 31-56 implement two for loop of Algorithm 2.8. In line 37 the sub-segment of oversized segment is computed by Delta_min class in Appendix A7 and this segment stores in delta_Min object. If the conditions in Lines 38-43 is satisfied, then the set of nodes will be added to the undersized segment (Lines 46-47) and removed from the oversized segment (Lines 48-56). If there are no adjacent oversized and undersized segments, then the **while** loop is terminated.

Implementation of trust node selection process (Block 9) includes 2 classes as follows:

Defining class of Bordering_Node_Details: Appendix A9 shows the Bordering_Node_Details class. Bordering node class checks all the list of the set of the bordering nodes for duplication. If there is no duplication, then the node will be added to the list (Lines 6-20)

Defining class of Trust_Node_Selection: Trust_Node_Selection class (Appendix A10) is used to perform the Algorithm 2.11 in Chapter 2. Lines 2-22 represent initialization part. Inputs of the function (Bordering_Node_Details) is the original graph (G) and updated segment collection (S). All the links belonging to the original graph are checked. If bordering node (x) and bordering node (y) are not equal, the intersegment property of the Edge property changes to true and the bordering node, (x, y), is added to the bordering node set (Lines 26-37). All the links are checked: if the intersegment feature is equal true, it is added to the intersegment link array (L_xy) (Lines 38-51). Line 56 is the linear programming problem solved by Matlab. This function returns the set of trust nodes.

3.2 Testing of Optimal TSP Program

In this section, we will compare the results of the application with results in Figures 9-14 in Chapter 2. Figure 26 shows the main page of the optimal TSP in smart grid SCADA networks.

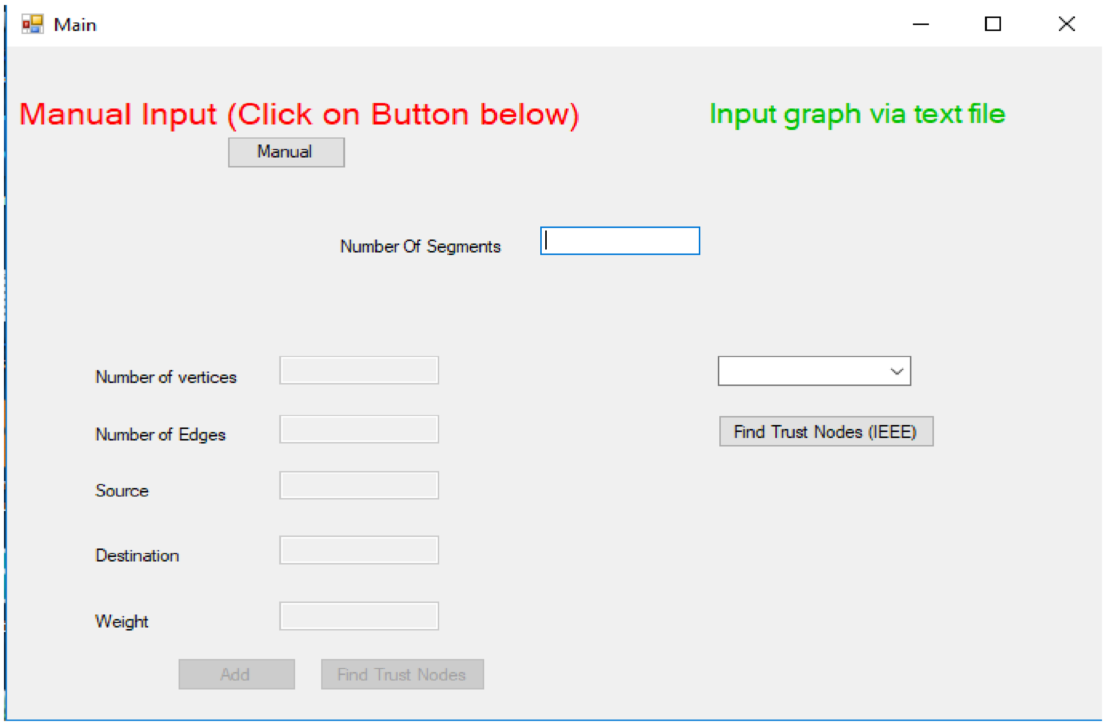


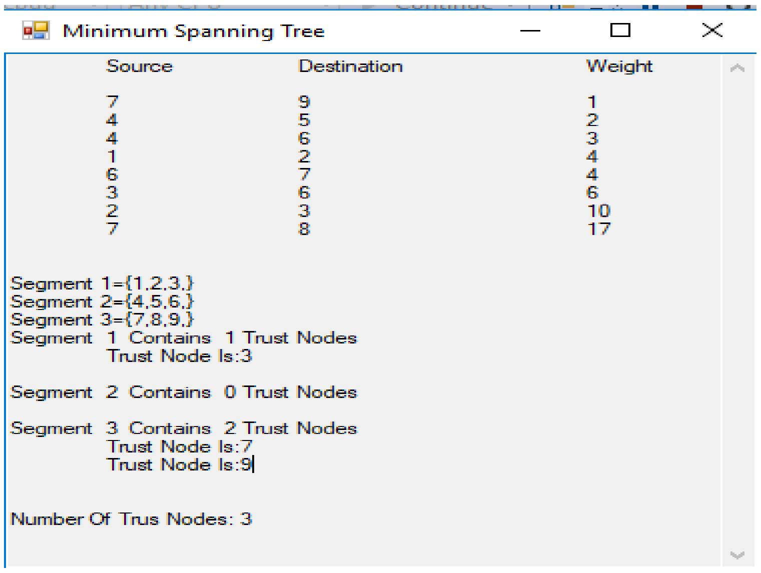
Figure 26: Main form of the program.

The default input graph is considered as text file. The comboBox property on the right side of the form shows the databases of the IEEE test system topologies. The number of segments is initialized by the textbox. The button “Find Trust Nodes IEEE”, executes the Algorithms 2.7-2.10, and the results will be shown in the Minimum Spanning Tree and Uniform forms.

In Figure 26, if user clicks on the button “Manual”, the input type will be changed to the manual input. Number of vertices textbox receives the number of nodes in the graph. Number of edges textbox receives the number of links of the graph. Source and Destination text boxes receive the node number of two connected nodes of the link. Weight textbox receives the propagation or distance between two nodes of the link. By pressing the button “ADD”, the new information of the link of the graph will be added to the Edge object (this object is defined in Edge class, Appendix A). The functionality of the Find Trust Nodes button is to execute the Algorithms 2.7-2.10.

The complete graph is shown in Figure 9. The data of this graph is used as input to the application.

Figure 27 demonstrates the minimum spanning tree and the trust systems in each segment (Figures 10 and 14). Minimum spanning tree form (Figure 27) at first does not have any contents. After computing the MST and segments the values will be changed to string type and written on the form.



```
            Source              Destination                 Weight

            7                   9                           1
            4                   5                           2
            4                   6                           3
            1                   2                           4
            6                   7                           4
            3                   6                           6
            2                   3                           10
            7                   8                           17

Segment 1={1,2,3,}
Segment 2={4,5,6,}
Segment 3={7,8,9,}
Segment  1  Contains  1 Trust Nodes
            Trust Node Is:3

Segment  2  Contains  0 Trust Nodes

Segment  3  Contains  2 Trust Nodes
            Trust Node Is:7
            Trust Node Is:9


Number Of Trus Nodes: 3
```

Figure 27: Minimum spanning tree form that shows the MST graph and trust nodes in each segment base on figure 8.

We test on Figures 9-14. The complete SCADA network is depicted in Figure 8. The links of minimum spanning tree of SCADA network in Figure 10 are (1, 2), (2, 3), (3, 6), (6, 4), (4, 5), (6, 7), (7, 8) and (7, 9) which are same as the links of minimum spanning tree in Figure 27. The number of segments K is 3. After information of the MST in Figure 27, the nodes of segments and trust systems in each segment are shown. Trust nodes in Figure 27 are 3, 7, 9 and segment 1 contains one trust node (3), segment 2 does not have any trust node and segment 3 has two trust nodes (7, 9). All the results are the same as in the Figure 14.

In Figure 14, we realized that one segment does not have trust node on the other hand, the total number of trust nodes is 3, and number of segments is 3, s1={1,2,3}, s2={4,5,6}, and s3={6,7,8}. As a consequence, we decide to distribute the trust nodes through the segments as much as possible. In the next Chapter, I am going to describe how uniformity of trust nodes distribution can improve the dispersion of trust nodes over network segments as much as possible.

3.3 Summary

In this chapter, we implemented optimal TSP problem. The problem is written by the Microsoft C#.net which is an objective oriented language. We defined classes for Edge, Graph, MST, LPP, Delta min, bordering node and trust selectin. System scheme of the optimal TSP problem in [1] is depicted in Figure 21. The input of the implemented program is divided into two types. First one is a manual input which user indicates the node numbers, propagation delay or weight of the links, links and number of segments by filling the form. In the second one, user selects one of the IEEE test system topologies and the software extract the information of selected topology from the related text file. The outputs of the software include, minimum spanning tree form that shows the MST graph details, segment sets and trust nodes in each segment. We tested the program by comparing the result of our program with the result in [1]. Details of the graph in Figure 14 are inserted manually to the program. By comparing the results, we notice that both of them are same.

Chapter 4

IMPROVEMENT OF THE OPTIMAL TSP UNIFORMITY

In this chapter, we describe how uniformity of trust nodes distribution can be improved. We propose the Uniformity algorithm to solve the optimal TSP uniformity. This chapter includes two sections. In Section 4.1, definition and implementation of Uniformity algorithm is discussed. In Section 4.2, the uniformity of trust systems distribution over network segments is compared versus uniformity of the original TSP [1].

4.1 Definition and Implementation of Uniformity of TS Distribution over Network Segments

Trust systems are installed to the smart grid networks to monitor and control the traffic packets to block the dispersion of malicious packets through the segments. In discussed optimal TSP problem security is optimized but the dispersion of trust systems was not considered. Consequently, segments may have quite different number of TS allocated to them, some may have many TS, other may have no one TS. Segments that are not equipped by TS may deteriorate its security and the segments that overloaded may cause delay on the segment.

To optimize the security and minimize the operational delay we propose to optimize uniformity of TSP allocation to the segments (Optimal TSP uniformity problem). This problem improves TSP which means that the number of TS will be exactly same as after optimal TSP problem solving, and all inter-segment links are connected to at least one TS. The number of TS and inter-segment limitation are considered as constraints for the optimal TSP uniformity problem.

We introduce the Uniformity optimization algorithm to solve the optimal TSP uniformity problem. The relative standard deviation, coefficient of variation, of TS number per segment is used as a metric to measure how much the segments are uniform in number of trust systems.

For example, trust nodes in Figure 14 are 3, 7, 9. Nodes 7, 9 are in one segment, s3; node 3 belongs to another segment, s1, and one segment, s2, has not any TS that may deteriorate its security (vulnerable to compromise of other segments). Uniformity algorithm re-distributes the TS over the segments. For instance, Uniformity algorithm selects node number 6 as a trust node that belongs to the segment, s2, instead of the node number 7 belonging to s3. As a result, all segments have the same number of TS and all inter-segment links are connected to at least one TS. In Figure 14 the coefficient of variation of trust node number per segment was 0.81 and in Figure 28, after executing the Uniformity optimization algorithm, this value changes to 0. As a result, we improved the dispersion of trust system by 81%. Figure 28 shows the result of the Uniformity algorithm.

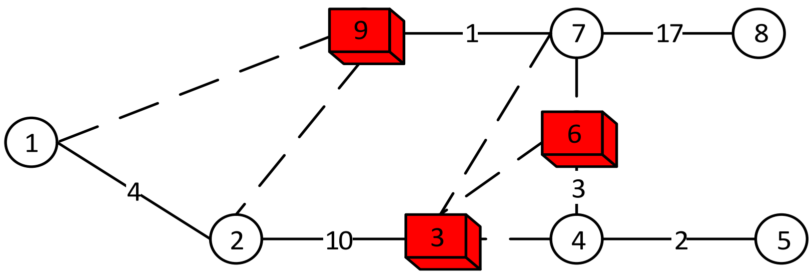


Figure 28: Distributing trust systems over segments by using Uniformity optimization algorithm, algorithm 4.1.

Algorithm 4.1 describes the Uniformity optimization algorithm to distribute the trust systems through the segments.

Algorithm 4.1: Pseudo code of Uniformity optimization algorithm.

```
Uniformity optimization
Input: S, $V^{Trust}$,Lss'.// S denotes the .set of segments, $V^{Trust}$ denotes the set of trust nodes and Lss' denotes the set of all inter-segment links, Lss'={e1,e2,..en} where e=(v1,v2) denotes the inter-segment link.
Output: $V^{Trust}$ // Updated trust nodes set.
1.  Begin
2.  Ns=$\left\lceil\frac{|V^{Trust}|}{|S|}\right\rceil$; O_cv=Computing the coefficient of variation of trust system// limitation of trust nodes in each segment. O_cv is CV of trust system
3.  balance=0; ph=0; $V^{Trust}_{temp} = \emptyset$; // $V^{Trust}_{temp}$ is a temporary set of trust nodes.
4.  while (balance<1) do
5.  ph=0; $V^{Trust}_{temp} = V^{Trust}$;// ph is a decision variable to terminate the while loop. $V^{Trust}_{temp}$ store the value of $V^{Trust}$.
6.  //searching to find overloaded and underloaded segments
7.      For i=1 to |S| do
8.          For j=1 to |S| do
9.                  If ((number of trust nodes in segment i<Ns) and
                    (number of trust node in segment j>Ns )) then
10.
11.                     X1=Find trust node in segment j adjacent to segment i;
```

```
12.         For all e ϵ Lss' do
13.             //searching to find the intersegment link between
                overloaded segment j and underloaded segment i
14.                 Y1=Find segment such that Lss'.e.v1 belongs
                    to it;
15.                 Y2=Find segment such that Lss'.e.v2 belongs
                    to it;
16.   //If there is an inter-segment link and one node of the link, hosted
      the trust system and other node does not then change the place of
      trust system.
17.             If((((Y1==segment i) and (Y2==segment j))
                or((Y1==segment j) and(Y2==segment j)) then
18.                 If ((X1==Lss'.e.v2) and(Lss'.e.v1$\notin$ $V^{Trust}$)))
                    then
19.                         $V^{Trust}$={$V^{Trust}$\X1};//Remove.
20.                         $V^{Trust}$={$V^{Trust}$ $\cup$Lss'.e.v1};//Add.
21.                 End if
22.             If ((X1==Lss'.e.v1) and(Lss'.e.v2$\notin$ $V^{Trust}$)))
                then
23.                         $V^{Trust}$={$V^{Trust}$\X1};//Remove.
24.                         $V^{Trust}$={$V^{Trust}$ $\cup$Lss'.e.v2};//Add.
25.             End if
26.
27.             End if
28.             ph++;
```

```
29.                 End for
30.                     For all e ϵ Lss' do// checking the constraint, inter-
                        segment links must be connected to at least one trust
                        nod,
31.                       If (Lss'.e.v1 $\notin V^{\mathrm{Trust}}$ ) and
32.                           (Lss'.e.v2 $\notin V^{\mathrm{Trust}}$) then
33.                               $V^{\mathrm{Trust}}=V^{\mathrm{Trust}}_{\mathrm{temp}}$;// It means that one inter-
                                  segment link is not connected to trust system
                                  as a result, the new trust nodes cannot accept
                                  and the value of $V^{\mathrm{Trust}}$ returns to $V^{\mathrm{Trust}}_{\mathrm{temp}}$,
                                  nothing change,
34.                       End if
35.                   End for
36.           End if
37.         End for
38.       End for
39.       If(ph==0) then
40.          Balance++;// all segments are checked
41.       End if
42.  End while
43.  P_cv=Compute the coefficient of variation of trust system after
     improvement
44.  Improve_Measure=O_cv – P_cv;// show how much the dispersion
     of trust system is improved.
45.  Return $V^{\mathrm{Trust}}$;
```

The inputs of the Algorithm 4.1 are set of segments, S, and set of trust nodes, V^{Trust}, and set of all inter-segment links. The output is the updated trust nodes set, V^{Trust}, which includes trust nodes that are distributed over segments. In line 2, the limitation of the size of the segments in number of trust nodes and coefficient of variation of trust system over segments before improving are calculated. In line 3 the $V_{\mathrm{temp}}^{\mathrm{Trust}}$ is a temporary set of trust nodes to store the value of V^{Trust} before it is updated. Line 4 is a while loop and the function of this loop is to find the overloaded and underloaded segments. If these segments are found, Line 9, then the adjacent trust node, which is connected to underloaded segment by the inter-segment link, in overloaded segment must be recognized, line 11. Line 12 is a local search loop and it finds the inter-segment link between oversized and undersized segments. Line 16 checks 2 conditions. First condition checks the nodes of inter-segment link that these nodes belong to underloaded and overloaded segments. Second condition checks that the node of the inter-segment link that is located in overloaded segment hosted the TS and the intersection of the other node with V^{Trust} is empty, it is not a trust node. If these conditions return true value, then the place of trust system is swapped between the nodes of the inter-segment link, lines 18-19. After finding the new node that hosted trust system the V^{Trust} is updated. The constraint of the problem is that all inter-segment links must be connected to at least one trust node. Lines 30-35 are used to check all inter-segment links for this reason. If even one inter-segment link is not connected to at least one trust node in the updated V^{Trust}, Lines 31-32, then the value of V^{Trust} returns to its value before updating, $V_{\mathrm{temp}}^{\mathrm{Trust}}$ .The search function to find the overloaded and underloaded segments has been continued since there is no overloaded and underloaded segments, line 39. In line 43 the coefficient of variation of TS over segments after improving is computed. In line 44, the improvement value is return. In line 45 the updated trust nodes set is returned. An example of Uniformity optimization algorithm work is given in Example 4.

Example 4: Application of Uniformity algorithm, Algorithm 4.1, to the graph in Figure 14.

Input: S={s1,s2,s3}={{1,2,3},{4,5,6},{7,8,9}}; $V^{\text{Trust}} = \{3,7,9\}$; Lss'={(1,9),(2,9),(3,6),(3,7),(3,4),(6,7)}.

Ns=$\left\lceil\frac{3}{3}\right\rceil = 1$; balance=0; $V_{\text{temp}}^{\text{Trust}} = \emptyset$; O_cv=0.81;

First iteration of while loop// balance=0<1

Ph=0; $V_{\text{temp}}^{\text{Trust}} = \{3,7,9\}$;

i=1;

j=1;

number of trust node in s(i)=s1=1 is not smaller than Ns=1 and number of trust node in s(j)=s1=1 is not greater than Ns=1; //both conditions is not satisfied.

i=1;

j=2;

number of trust node in s(i)=s1=1 is not smaller than Ns=1 and number of trust node in s(j)=s2=0 is not greater than Ns;//both conditions is not satisfied

i=1;

j=3;

number of trust node in s(i)=s1=1 is not smaller than Ns=1 and number of trust node in s(j)=s3=2 is greater than Ns;// one condition is not satisfied.

i=2;

j=1;

number of trust node in s(i)=s2=0 is smaller than Ns=1 and number of trust node in s(j)=s1=1 is not greater than Ns;// one condition is not satisfied.

i=2;

j=2;

number of trust node in s(i)=s2=0 is smaller than Ns=1 and number of trust node in s(j)=s2=0 is not greater than Ns;// both conditions are not satisfied.

i=2;

j=3;

number of trust node in s(i)=s2=0 is smaller than Ns=1 and number of trust node in s(j)=s3=2 is greater than Ns.

X1=7;// adjacent trust node in oversized segment, s3, that is connected to undersized segment, s2, through the inter-segment link.

Ls2s3={(6,7)};//inter-segment link between segments s2 and s3.

Y1=s2; Y2=s3;

Ls2s3.e.v2=7=X1;

Ls2s3.e.v1=$6\cap\{3,7,9\}=\emptyset$;

$V^{\mathrm{Trust}} = \{3,6,9\}$;

Ph=1;

// Checking all inter-segment links that are connected to at least one trust node as follows;

Lss'={e1,e2,e3,e4,e5,e6}={(1,9),(2,9),(3,6),(3,7),(3,4),(6,7)}

e1.v1=1 does not belong to V^{Trust} e1.v2=9 belongs to V^{Trust};

e12.v1=2 does not belong to V^{Trust} e2.v2=9 belongs to V^{Trust};

e3.v1=3 belongs to V^{Trust} e3.v2=6 belongs to V^{Trust};

e4.v1=3 belongs to V^{Trust} e1.v2=7 does not belong to V^{Trust};

e5.v1=3 belongs to V^{Trust} e5.v2=4 does not belong to V^{Trust};

e6.v1=6 belongs to V^{Trust} e6.v2=7 does not belong to V^{Trust};

//All inter-segment links are connected to at least one trust node as a consequence updated set of trust nodes , V^{Trust}, is accepted. V^{Trust}={3,6,9};

i=3;

j=1;

number of trust node in s(i)=s3=1 is not smaller than Ns=1 and number of trust node in s(j)=s1=1 is not greater than Ns;// both conditions are not satisfied.

i=3;

j=2;

number of trust node in s(i)=s3=1 is not smaller than Ns=1 and number of trust node in s(j)=s1=1 is not greater than Ns;// both conditions are not satisfied.

i=3;

j=3;

number of trust node in s(i)=s3=1 is not smaller than Ns=1 and number of trust node in s(j)=s1=1 is not greater than Ns;// both conditions are not satisfied.

Ph=1 then balance=0

//in the second iteration of while loop the value of ph does not change, ph=0, because there is no oversized and undersized segment.

Ph=0 then balance=1;//the while loop is terminated

The updated set of trust node, uniform trust nodes, is returned by the Uniformity algorithm.

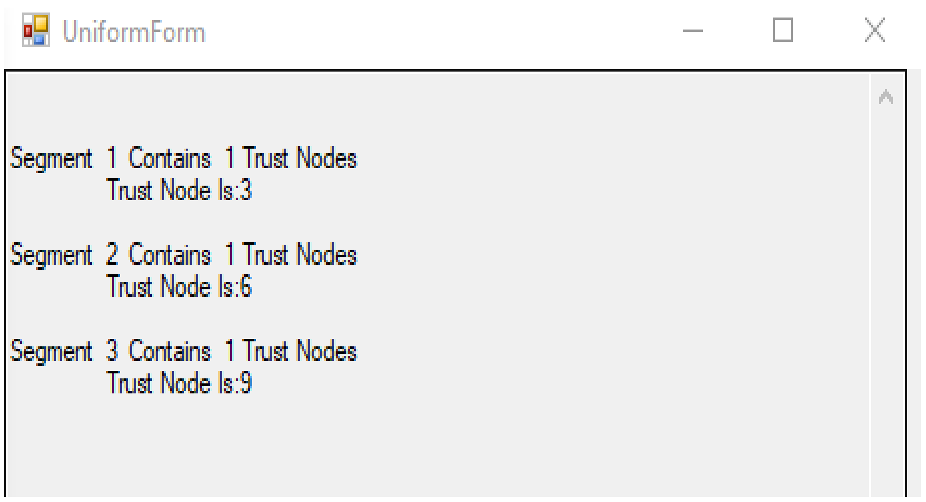
P_cv=0;

Improve_Measure=0.81-0=0.81//improved by 81%;

Output: $V^{Trust} = \{3,6,9\}$;

Appendix A11 shows the codes that make the uniform distribution of trust nodes over the segments (uniformity algorithm). In uniformity algorithm, number of trust nodes does not change but the place of the trust node may be changed. This algorithm has a loop that checks the number of trust system in segment set to find the oversized set (Lines 16-48). It checks the other side of the link of the trust node to check that at first this node belongs to undersized trust system in bordering node set and latter checks that the other side of the link is not a trust node (Lines 49-57). If all the conditions are satisfied, then the trust node is removed from the oversized segment and is added to an undersized set (Lines 59-66). This loop continues until all the trust nodes are checked.

Figure 29 shows the UniformForm, that improves the optimal TSP described in Chapter 2 and shown in Figure 14. It is clear that trust system moves from node 7 to node 6. As a result, each segment has one trust node.



```
Segment  1  Contains  1 Trust Nodes
         Trust Node Is:3

Segment  2  Contains  1 Trust Nodes
         Trust Node Is:6

Segment  3  Contains  1 Trust Nodes
         Trust Node Is:9
```

Figure 29: Uniform Form that shows the trust nodes after uniformity problem solving for the system on figure 14.

4.2 Testing Results

Figure 30 depicts a bar chart. This bar chart compares the number of trust nodes in each segment (Figure 14) with the number of trust nodes in each segment after uniformity improvement (Figure 27). It is clear that each segment contains one trust system and the segments are more balanced in quantity of trust systems.

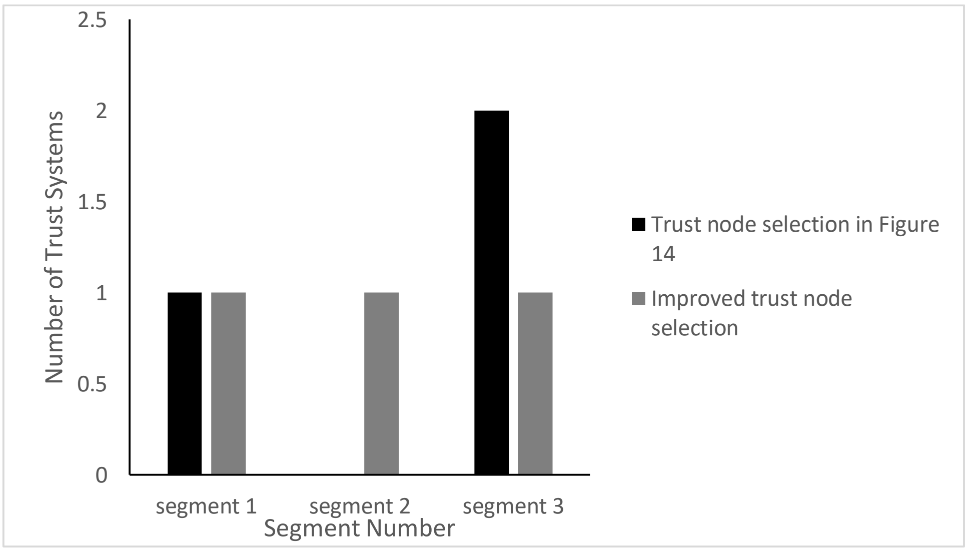


Figure 30: Number of trust systems in each segment.

The relative standard deviation is used as a metric to measure the dispersion of trust nodes over the segments. Figure 31 shows the coefficient of variation of trust nodes through the segments. The smaller value indicates that the dispersion of trust nodes is more balance. In Figure 31, the coefficient of variation of trust nodes for the SCADA network in Figure 14 is 0.81 and the coefficient of variation of trust systems after executing the uniformity (Figure 29) is 0, it means that the segments are completely balanced in quantity of trust nodes and each segment has the same quantity of trust nodes (each segment contains 1 trust node).

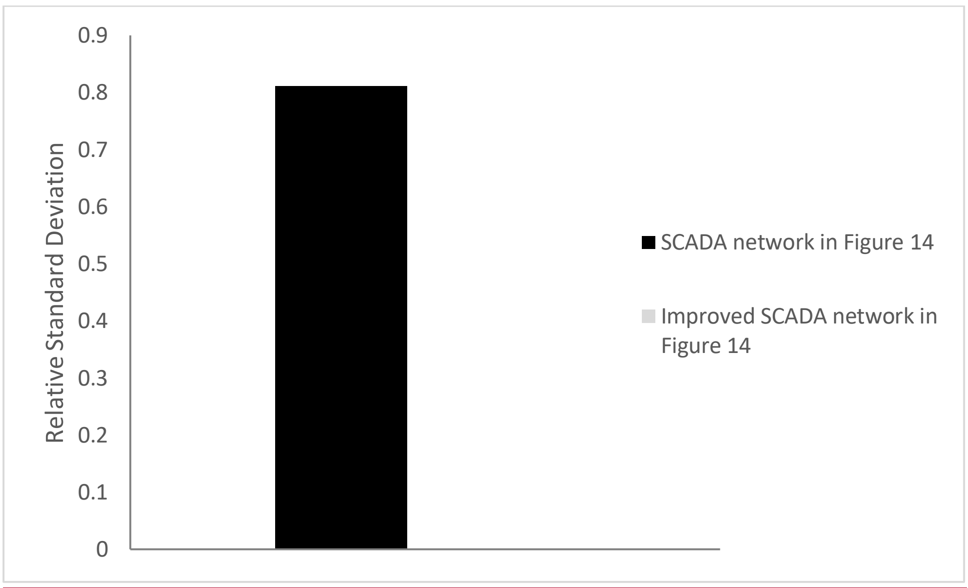


Figure 31: Relative standard deviation of trust system number for 3 segments of SCADA system in figure 14. The bar for improved version is not shown as equal to zero.

4.3 Summary

In this chapter, improvement of optimal TSP was discussed. The idea comes from Figure 14. When we analyzed the Figure 14, we realized that one segment does not have any TS and instead of this one segment has 2 TS and another one has 1 TS. This problem improves TSP which means that the number of TS will be exactly same as after optimal TSP problem solving, and all inter-segment links are connected to at least one TS. The number of TS and inter-segment limitation are considered as constraints for the optimal TSP uniformity problem. in such a way that all inter-segment links connected to at least one trust node and number of TS. We proposed Uniformity optimization algorithm to distribute the trust systems over the segments. By testing the algorithm, graph in Figure 9 used as input, and comparing the results with Figure 14 we realized that all segments were balanced in number of trust systems. The coefficient of variation was used to measure the dispersion of trust systems over the segments. Before performing the Uniformity algorithm, the coefficient of variation was 0.81 and after that it changed to 0, lower value of coefficient of variation means that segments are more balanced. As a consequence, we improved the dispersion of trust systems over the segments for the graph in Figure 9 by 81%.

Chapter 5

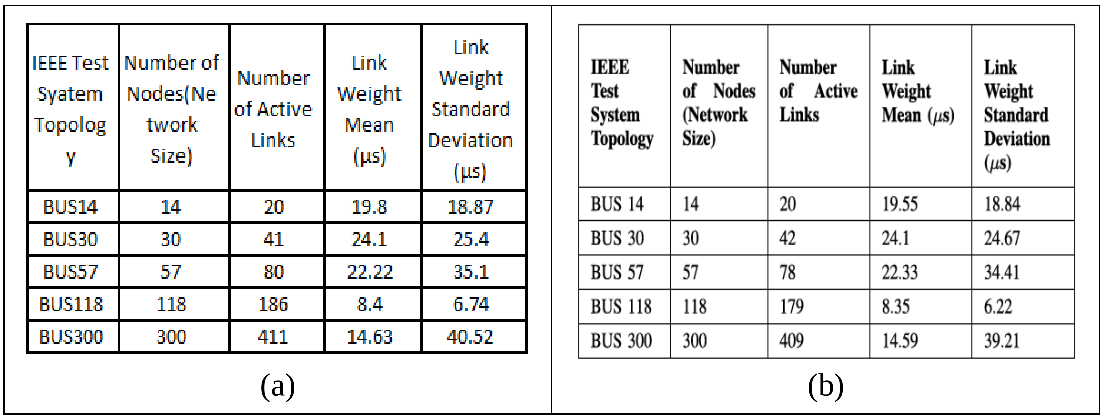
EXPERIMENTS ON IEEE TEST SYSTEM TOPOLOGIES

In this chapter, we will compare the results of our software for large and small networks with the numerical results shown in Chapter 2, Figures 18-20. Large networks contain BUS 118 and BUS 300 and BUS 14, BUS 30 and BUS 57 are members of small networks. The information of the small networks and large networks are used as a text file in the application Appendix B. Small networks are split into 3 to 6 segments with increment of 1. Large networks are divided into 5 to 30 segments with increments of 5. All experiments are run on a laptop with Intel core i7 2.10 GHz and 8GB RAM. We run the application one time for each IEEE test system topologies, because the input data are not changed. The numerical results are shown in Appendix C.

The obtained results for BUS14 are shown in Appendix C1, Figures (41-44). The obtained results for BUS30 are depicted in Appendix C2, Figures (45-49) and these results are shown in Figures (50-54) for BUS57. These numerical results include: number of TS, link mean weight, link weight standard deviation, average MST weight, average segment size and coefficient of variation of computed segment sizes. Figures (55-60) depicts the results for BUS118 and Figures (61-66) shows the experimental results for BUS300.

5.1 Experimental Results on Original TSP

Figure 32 compares the overview of experimental parameters. Figure 32(a) is a table of summary of experimental parameters on our experiment. Figure 32(b) shows a table of summary of experimental parameters in [1]. In comparison, the value of elements for BUS 30, BUS 57, BUS 118, BUS 300 are slightly different. I used the same databases and implemented the exact algorithms in [1]. These differences may be happened because the databases are updated. The differences on number of active links might affect on the MST, segments and number of trust nodes. The number of active links for BUS14 is the same in both Figures 32(a) and 29(b). As a consequence, BUS14 is a measure to compare our results with the results in [1].



IEEE Test Syatem Topolog y	Number of Nodes(Ne twork Size)	Number of Active Links	Link Weight Mean (µs)	Link Weight Standard Deviation (µs)
BUS14	14	20	19.8	18.87
BUS30	30	41	24.1	25.4
BUS57	57	80	22.22	35.1
BUS118	118	186	8.4	6.74
BUS300	300	411	14.63	40.52

(a)

IEEE Test System Topology	Number of Nodes (Network Size)	Number of Active Links	Link Weight Mean (μs)	Link Weight Standard Deviation (μs)
BUS 14	14	20	19.55	18.84
BUS 30	30	42	24.1	24.67
BUS 57	57	78	22.33	34.41
BUS 118	118	179	8.35	6.22
BUS 300	300	409	14.59	39.21

(b)

Figure 32: Comparison of the summary of experimental parameters in (a) our application. (b) in [1].

All experimental results are shown in Appendixes C1-C5. I insert our results in the Microsoft Excel to draw the bar charts and line charts. Figures 33-38 compare our experimental results with the results in [1].

Figure 33 compares the relative standard deviation of the calculated segment sizes in [1] (Figure 33(a)) with the relative standard deviation of the calculated segment sizes on our experiments. The trend of the Figure 33(b) is same as the trend of the Figure

33(a) for BUS14, BUS30 and BUS57 SCADA networks with 3 and 4 segments and it is same as the trend of the Figure 33(a) for BUS30 SCADA network with 5 and 6 segments.

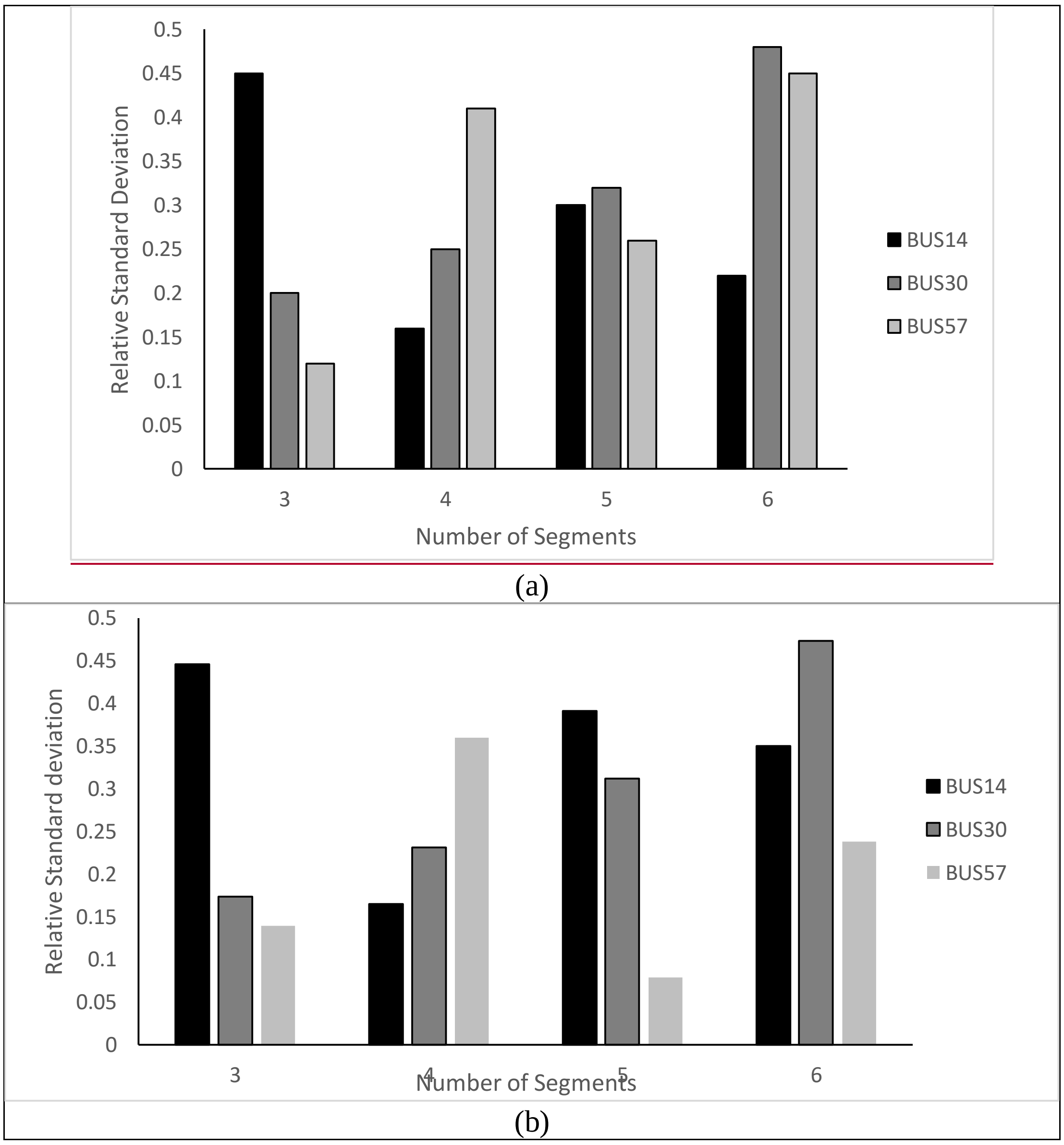
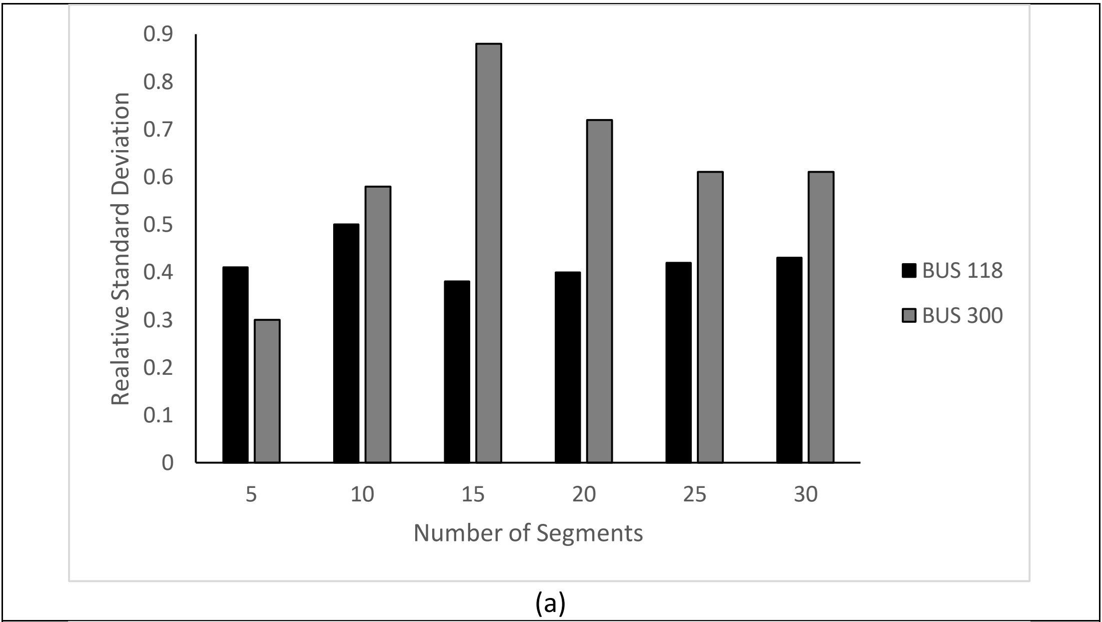


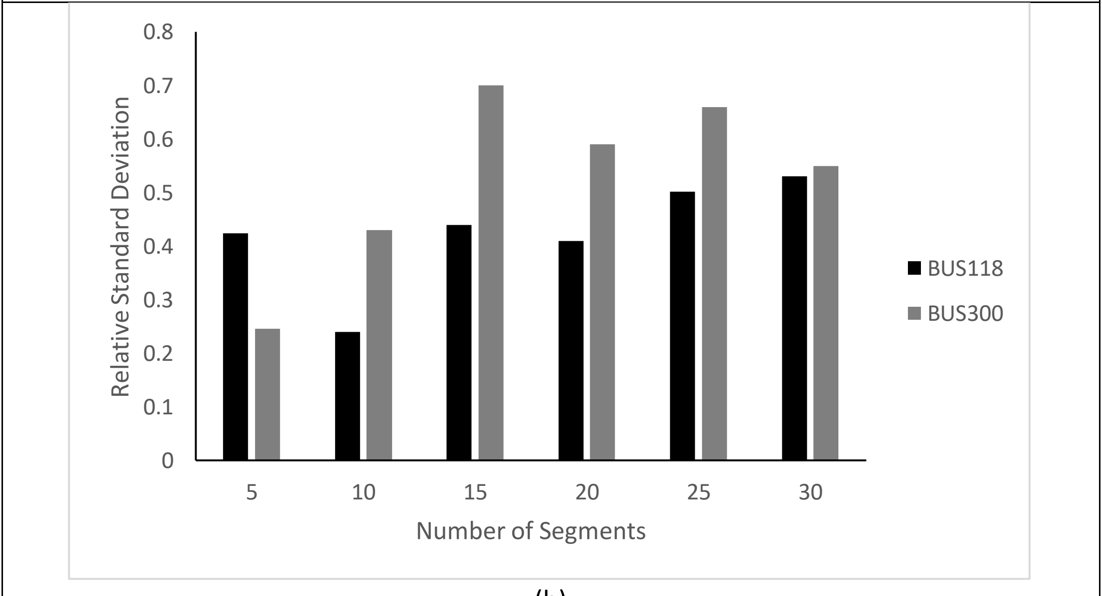
Figure 33: Relative standard deviation of the calculated segments sizes for small networks in (a) [1]. (b) our experiments.

Figure 34 compares the relative standard deviation of the calculated segment sizes in [1] (Figure 34(a)) with the relative standard deviation of the calculated segment sizes on our experiments. The trend of the Figure 34(b) is same as the trend of the Figure

34(a) for large BUS300, with 5, 10, 15, and 20 segments and the relative standard deviation of computed segment sizes for BUS118 is slightly different by comparing with the result in [1](Figure 34(a)).



(a)

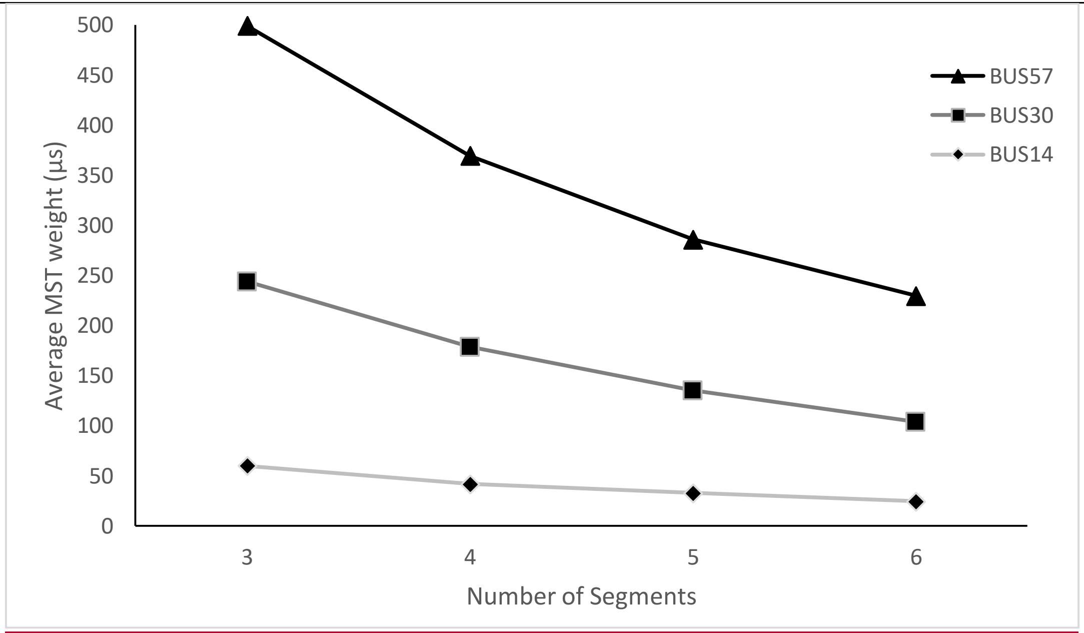


(b)

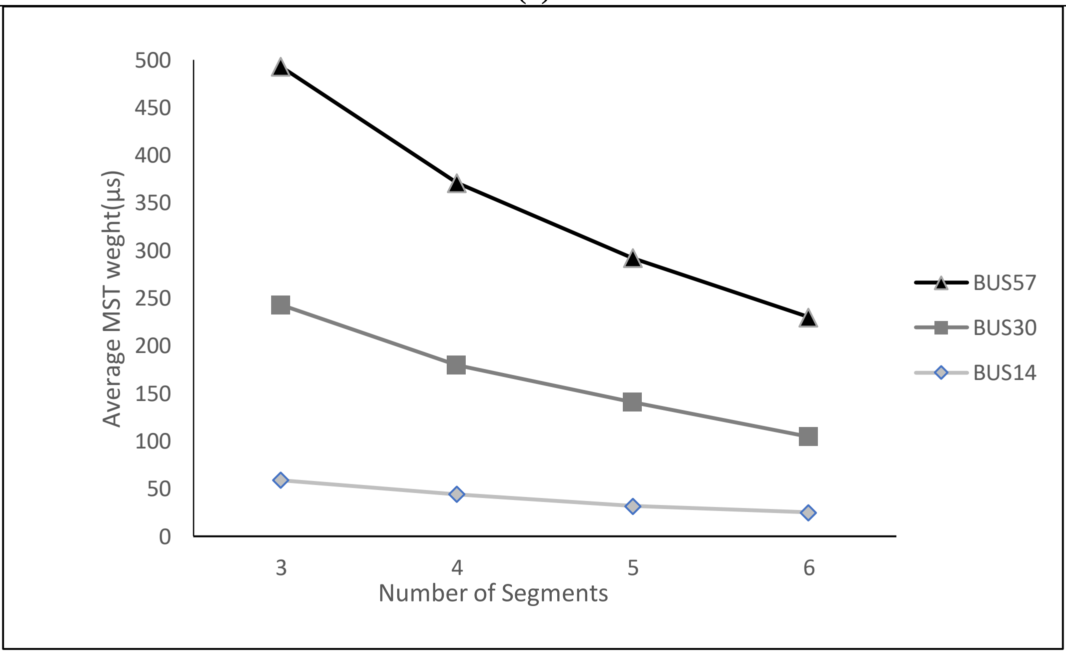
Figure 34: Relative standard deviation of the calculated segments sizes for large networks in (a) [1], (b) our experiments.

Figure 35 demonstrates the line chart for average of MST weights for small networks. Figure 35(a) depicts the average of MST weights for small networks in [1].

Figure 35(b) illustrates the line chart for average of MST weights for small networks on our experiments. In general, Figure 34(b) in comparison with Figure 34(a), the average of MST weight follows the same decreasing trend by the number of segments rises.



(a)

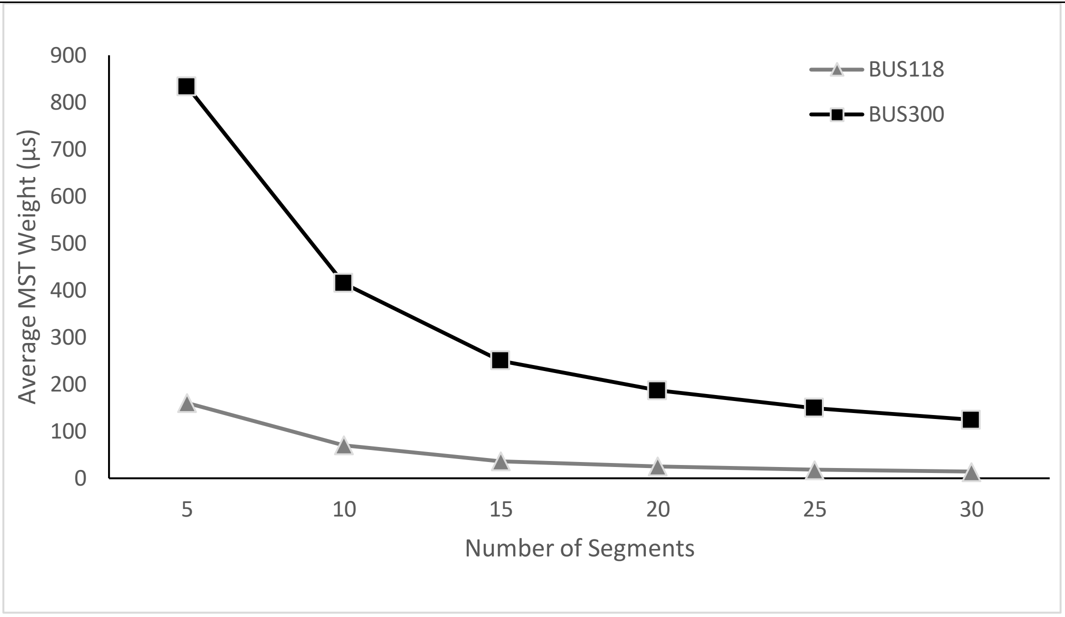


(b)

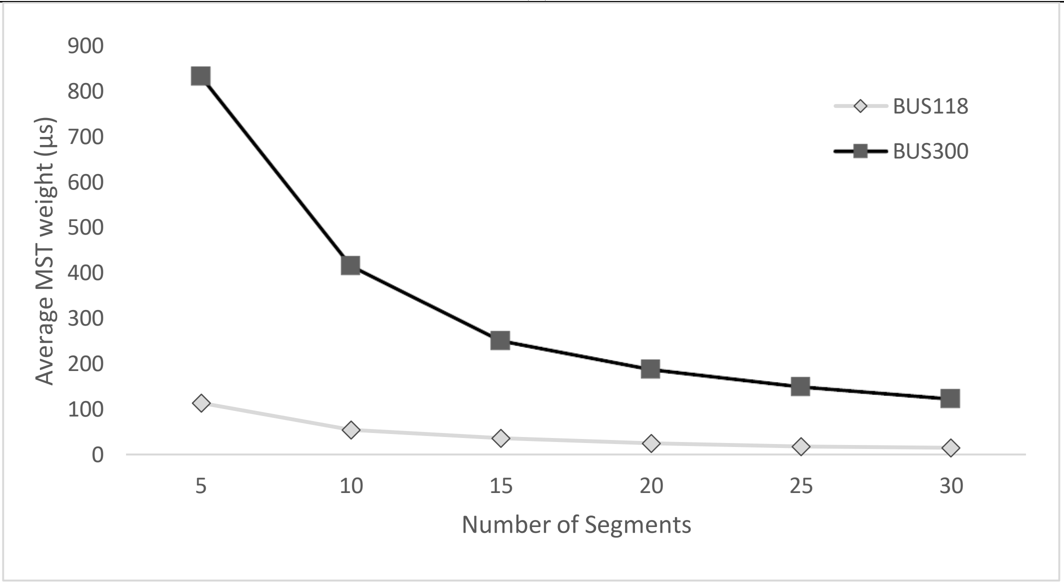
Figure 35: The average MST weights for small networks in (a) [1]. (b) our experiments.

Figure 35 and 36 demonstrates the line chart for average of MST weights for small networks. Figure 35(a) depicts the average of MST weights for small networks in [1]. Figure 35(b) illustrates the line chart for average of MST weights for small networks on our experiments. In general, Figure 35(b) in comparison with Figure 35(a), the average of MST weight follows the same decreasing trend by the number of segments rises.

Figure 36 depicts the line chart for average of MST weights for large networks. Figure 36(a) illustrates the average of MST weights for large networks in [1]. Figure 36(b) illustrates the line chart for average of MST weights for large networks on our experiments. In general, Figure 36(b) in comparison with Figure 36(a), the average of MST weight follows the same decreasing trend by the number of segments rises.



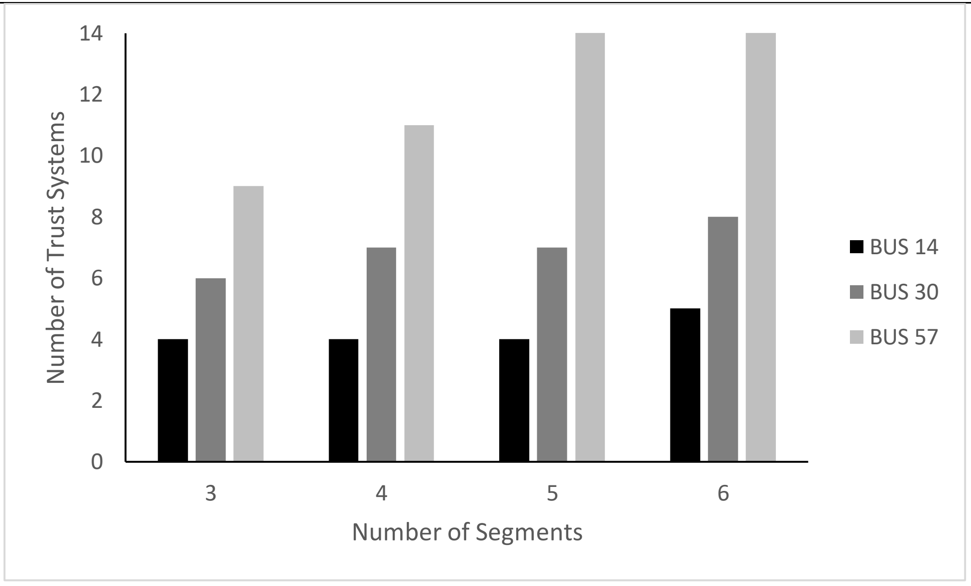
(a)



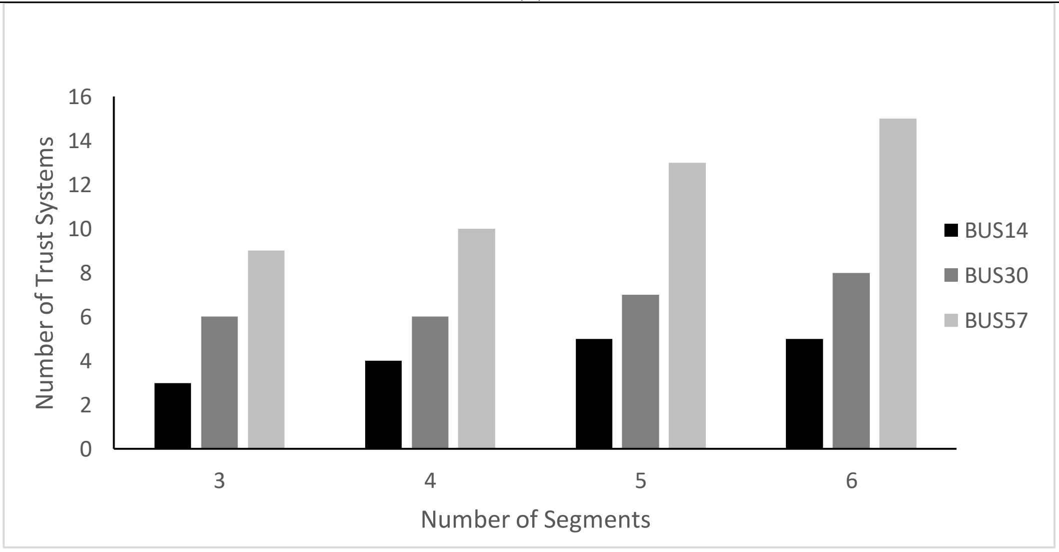
(b)

Figure 36: The average MST weights for large networks in (a) [1], (b) our experiments.

Figure 37 indicates the bar chart for required number of trust system related to the number of segments for small networks. Figure 37(a) represents the required number of trust system related to the number of segments in [1]. Figure 37(b) shows the bar chart for required number of trust system related to the number of segments on our experiment. There is a slight difference between Figure 37(a) and Figure 37(b), and this is because of the differences of our database with database used in [1]. The trend of the bar chart in Figures 37(a) is same as the trend of the bar chart in Figure 37(b). Both bar charts in Figure 37 follows an increasing trend as the number of segments increases and also the number of trust systems increases as the SCADA networks size rises.



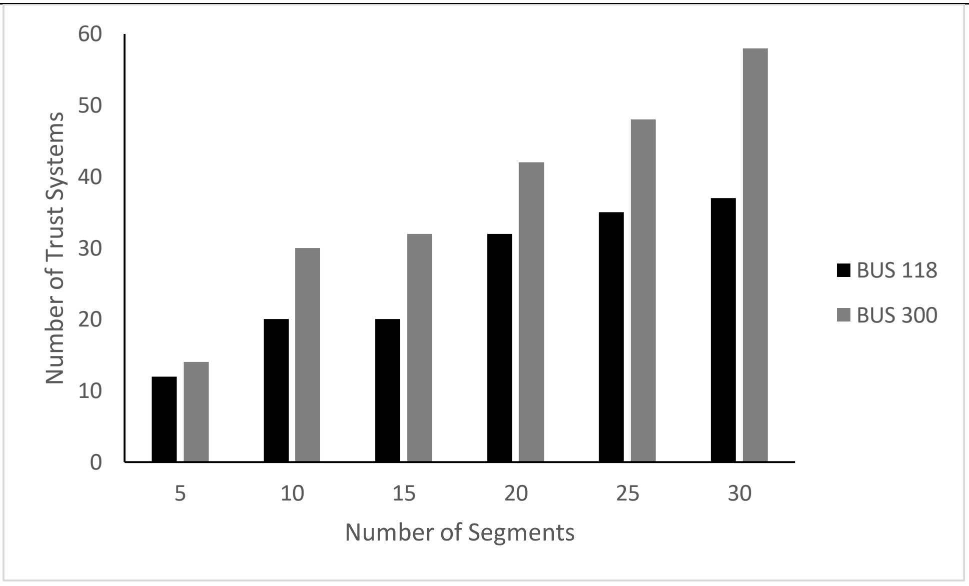
(a)



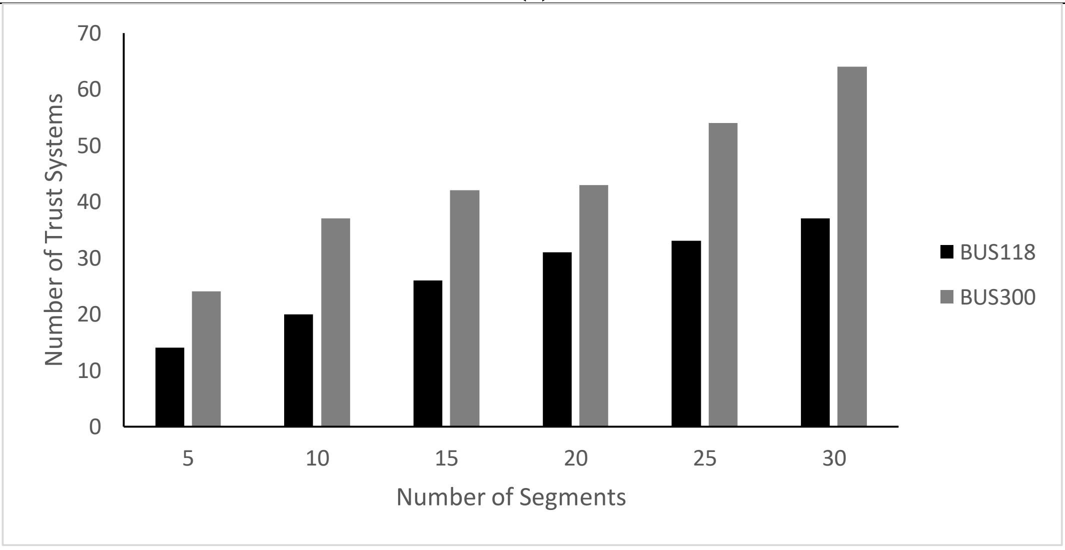
(b)

Figure 37: Required quantity of trust systems related to the quantity of segment for small networks in (a) [1]. (b) our experiments.

Figure 38 displays the bar chart for required number of trust system related to the number of segments for large networks. Figure 38(a) represents the required number of trust system related to the number of segments in [1]. Figure 38(b) shows the bar chart for required number of trust system related to the number of segments on our experiment. The tendency of bar chart in Figure 38(b) is same as the tendency in Figure 38(a) but the required number of trust system is slightly different. it is happened because of the difference in number of links and link mean weigh as mentioned at the beginning of the chapter.



(a)



(b)

Figure 38: Required quantity of trust systems related to the quantity of segment for large networks in (a) [1]. (b) our experiments.

5.2 Experimental Results on Comparison of Original and Proposed TSP

Figure 39 represents coefficient of variation, relative standard deviation, of dispersion of the trust systems through the segments. BUS14 in line chart is related to the coefficient of variation of dispersion of the trust nodes through the segments without uniformity improvement algorithm for BUS14. BUS14 proposed in line chart is related to the coefficient of variation of dispersion of the trust nodes through the segments with uniformity improvement algorithm for BUS14. The proposed BUS14 (improved by uniformity) exhibits lower coefficient of variation than BUS14. Segments of proposed BUS14 in number of trust systems are more balanced compare to BUS14 without uniformity in [1]. The value of coefficient of variation for BUS14 is 1 and this value for proposed BUS14 is 0 with 3 and 5 segments. As a consequence, the dispersion of the trust nodes (balancing segment in number of trust nodes) is improved by 100% for BUS14 with 3 segments. Figure 39 shows that dispersion of the trust nodes is improved by 81% with 4 segments and is improved by 35% with 6 segments.

Coefficient of variation of the size of the segments of trust systems in BUS14 is calculated manually and Excel is used to show the line chart. We just test the Uniformity optimization algorithm on BUS14 with 3, 4, 5 and 6 segments. Example 5 shows the calculation of the coefficient of variation of TS over the segments for BUS14 with 3 segments.

Example 5: Calculation of the coefficient of variation of TS over 3-segments for BUS14.

Number of TS over segments is 3;

The value of CV, standard deviation and average before executing Uniformity optimization algorithm are as follows:

Segment 1 contains 0 TS, segment 2 contains 1 TS and segment 3 contains 2 TS.

$\mu = \frac{0+1+2}{3} = 1;$

$$\sigma = \sqrt{\frac{(0-1)^2 + (1-1)^2 + (2-1)^2}{3-1}} = 1;$$

CV$= \frac{\sigma}{\mu} = 1;$// Coefficient of variation of TS over segments before Uniformity.

The value of CV, standard deviation and average after executing Uniformity optimization algorithm are as follows:

Segment 1 contains 1 TS, segment 2 contains 1 TS and segment 3 contains 1 TS.

$\mu = \frac{1+1+1}{3} = 1;$

$$\sigma = \sqrt{\frac{(1-1)^2 + (1-1)^2 + (1-1)^2}{3-1}} = 0;$$

CV$= \frac{\sigma}{\mu} = 0;$// Coefficient of variation of TS over segments after Uniformity.

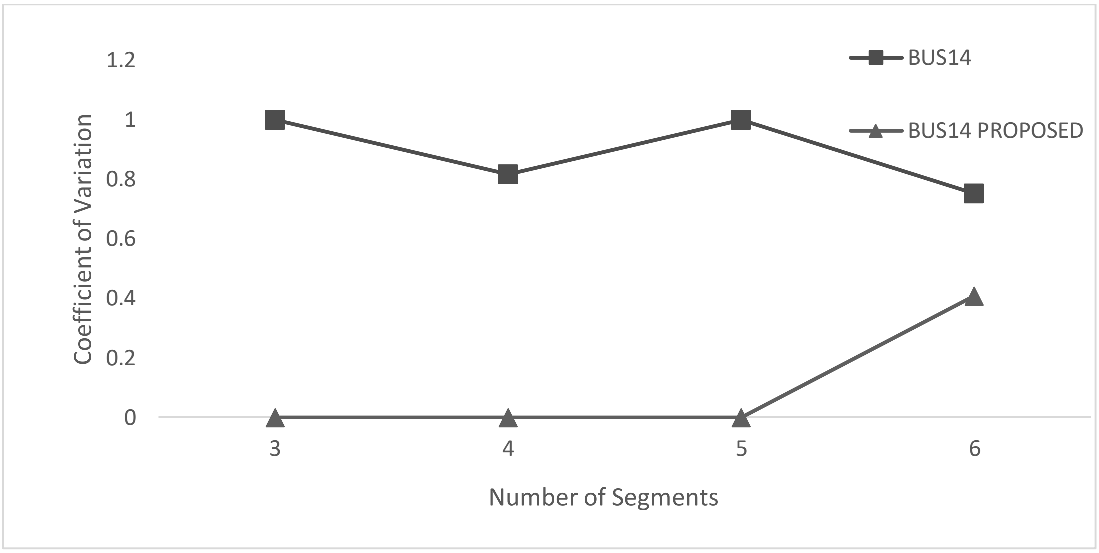


Figure 39: The coefficient of variation of the size of the segments of trust systems in BUS14.

Figure 40 shows the number of trust nodes in each segment of BUS14 (divided into 3 segments). The number of trust nodes in each segment of proposed BUS14 is equal to 1. It means that segments are completely balanced in number of trust nodes. In comparison, the number of trust nodes in each segment of BUS14 is different and follows an increasing trend (segment 1 contains 0 trust node, segment 2 contains 1 trust node and segment 3 contains 2 trust nodes).

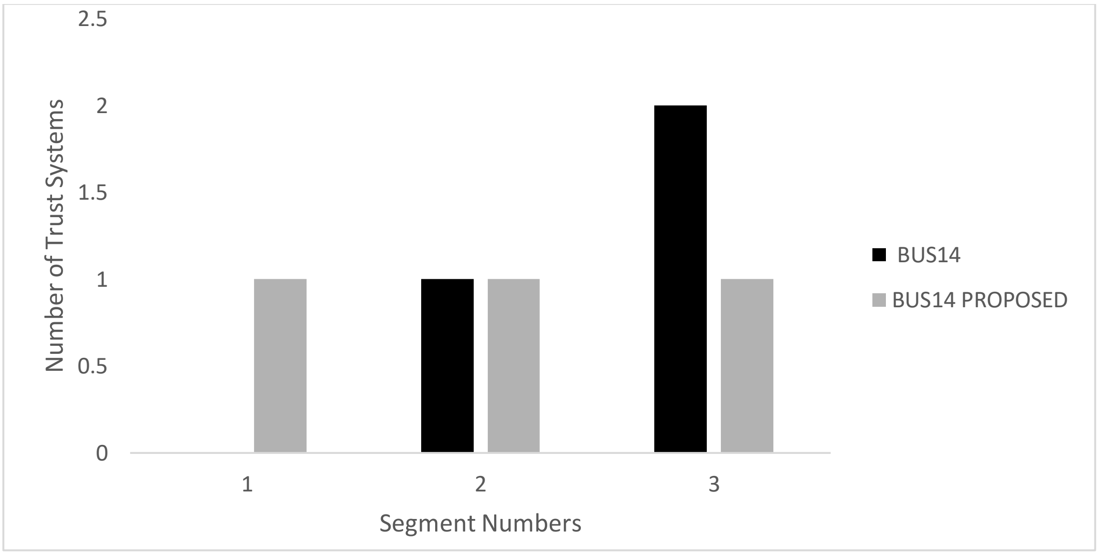


Figure 40: Number of trust systems in each segment of BUS 14 divided into 3-segments partitioning.

5.3 Summary

In this chapter we compared our outcomes with the numerical results in Chapter 2. The databases include five IEEE test system topologies for power grid systems that are divided into two groups of small networks, and large networks. The trend of the bar charts and line charts were same as the charts in Chapter 2. In general, by increasing the number of segments the required number of trust system increases and the mean value of MST weights decreases for both small and large networks. The segments in small networks are more balanced than the segments in large networks in size of the segments.

The coefficient of variation of the size of the segments of trust systems in BUS14 were considered to compare the results of the proposed TSP, Uniformity, with the original TSP. the coefficient of variation is 1 for BUS14 with 3 and 5 segments and this value is changed to 0 when the proposed TSP is performed. As a consequence, the dispersion of trust system is improved by 100% for BUS14 with 3 and 5 segments. When the BUS14 was divided into 4 and 6 segments the value of coefficient of variation is 0.81, divided into 4 segments, and it is 0.75, divided into 6 segments. After uniformity, these values were changed to 0 and 0.40. It means that, the dispersion of trust systems is improved by 81% and 35% when BUS14 divided into 4 and 6 segments. In general, the segments are more balanced in number of trust system by performing proposed TSP.

Chapter 6

CONCLUSION

The problem of optimal Trust System Placement (TSP) in SCADA networks is considered in the thesis. At present, as SCADA networks are connected to the internet the scope of cyber-security concerns becomes much wider. Trust systems are used to detect and block malicious activities. The nodes in SCADA networks that host the trust systems are known as trust nodes. Trust system consist of hardware and software agents and are expensive to deploy. The optimal TSP problem is considered to minimize the cost and maximize the security of the networks by installing minimum number of trust systems into the networks. The main part of the optimal TSP problem is segmentation. We divide the network into small networks, large networks are more vulnerable for intruders, and locate the trust system on the bordering node in such a way that all inter-segment links are connected to at least one trust node. We have implemented the TSP problem solving method proposed in [1].

We compare our experimental results on IEEE test system topologies with the results in [1] and we obtained the same results as in [1]. The results show us that by increasing the number of segments the required number of trust systems increases. The coefficient of variation is used to measure the uniformity of segments in size of number of nodes. We noticed that the small networks are more balanced than the large networks.

The trust systems number per segments was noted not uniform over the segments after optimal TSP problem solving. It may deteriorate security for the segments that are not equipped by trust systems and may increase the operational delay over the segments and inter-segment links that are oversized in number of segments. We propose the optimal TSP uniformity problem to maximize the uniformity of segments in number of trust systems and minimize the operational expenditure without change of trust nodes number and TS cover all inter-segment links. We proposed algorithm to solve the optimal TSP uniformity

The coefficient of variation of trust systems is used to measure uniformity of TSP. In original TSP problem the coefficient of variations for BUS14 with 3,4,5 and 6 segments are 1, 0.81, 1 and 0.75 in row, and after performing the Uniformity optimization algorithm the coefficient of variations are change to 0, 0, 0 and 0.40 with 3, 4, 5, and 6 segments. As a consequence, distribution of trust systems through the segment in comparison in [1], is improved by 100% (number of trust systems in each segment are same) with 3 segments and 5 segments and it is improved by 81% with 4 segments and it is improved by 35% with 6 segments.

REFERENCES

[1] MD. M. Hasan, H. T. Mouftah, “Optimal trust system placement in smart grid SCADA networks”, IEEE Sensors Journal, vol. 4, pp. 2907-2919, June 2016

[2] G. M. Coates “A trust system architecture for SCADA network security," IEEE Trans. Power Del., vol. 25, no. 1, pp. 158-169, Jan. 2010.

[3] J. Gonzalez ``Optimization of trust system placement for power grid security and compartmentalization," IEEE Trans. Power Syst., vol. 26, no. 2, pp. 550-563, May 2011.

[4] Y. Zhang, L. Wang, and W. Sun, ``Trust system design optimization in smart grid network infrastructure," IEEE Trans. Smart Grid, vol. 4, no. 1, pp. 184-195, Mar. 2013.

[5] A.-H. Mohsenian-Rad and A. Leon-Garcia, ``Distributed Internet-based load altering attacks against smart power grids," IEEE Trans. Smart Grid, vol. 2, no. 4, pp. 667-674, Dec. 2011.

[6] Black out report. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/2003_Italy_blackout

[7] Y. W. Law, M. Palaniswami, G. Kounga, and A. Lo, ``WAKE: Key management scheme for wide-area measurement systems in smart grid," IEEE Commun. Mag., vol. 51, no. 1, pp. 34-41, Jan. 2013.

[8] Public key infrastructure. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Public_key_infrastructure

[9] X-509. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/X.509

[10] M. M. Hassan and H. T. Mouftah, "A Study of Resource-Constrained Cyber Security Planning for Smart Grid Networks", In Proceedings of the EPEC2016, Ottawa, ON, Canada, pp. 1-6, 2016.

[11] M. M. Hassan and H. T. Mouftah, "Latency-Aware Segmentation and Trust System Placement in Smart Grid SCADA Networks", In Proceedings of the CAMAD2016, Toronto, ON, Canada, pp. 37-42, 2016.

[12] M. M. Hassan and H. T. Mouftah, "Optimization of Trust Node Assignment for securing Routes in Smart Grid SCADA Networks", IEEE Systems Journal, vol. 1, pp. 1-9, Sep 2018.

[13] Sorting algorithms. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Sorting_algorithm

[14] Bubble sorting algorithms. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Bubble_sort

[15] Disjoint-set Algorithm (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Disjoint-set_data_structure

[16] M. Laszlo and S. Mukherjee, ``Minimum spanning tree partitioning algorithm for microaggregation," IEEE Trans. Knowl. Data Eng., vol. 17, no. 7, pp. 902-911, Jul. 2005.

[17] Kruskal Algorithm (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[19] V. Chvatal, Linear Programming. New York, NY, USA: Freeman, 1983.

[20] University of Washington, Seattle, WA, USA. (2016). Power Systems Test Case Archive. [Online]. Available: http://www.ee.washington.edu/research/pstca/.

[21] Electrical grid. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Electrical_grid.

[22] Electric generator. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Electric_generator.

[23] Electric power transmission. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Electric_power_transmission.

[24] Electric power distribution. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Electric_power_distribution.

[25] Electrical load. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Electrical_load.

[26] Slack bus. (14 February 2019), Retrieved from https://en.wikipedia.org/wiki/Slack_bus.

[27] Coefficient of variation. (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Coefficient_of_variation

[28] Standard deviation (9 October 2018), Retrieved from https://en.wikipedia.org/wiki/Standard_deviation.

[29] Switch statement (9 October 2018), Retrieved from https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/switch

[30] File.ReadAllLines method (9 October 2018), Retrieved from https://docs.microsoft.com/en-us/dotnet/api/system.io.file.readalllines?view=netframework-4.7.2

APPENDICES

Appendix A: Source Codes

The source codes of the program are shown as follows.

Appendix A1: Source Code of Class Edge:

```
class Edge
{
public int Source;
public int Destination;
public int Weight;
public int minDegree=0;
public int maxdeg = 0;
public decimal normalizeWeight;
public decimal we=0;
public bool Active = false;
public bool delta_Seg;
public bool Intersegment = false;



//Insert edge
public  void add_Edge(int src, int dst , int wgth)
{
Source = src;
Destination = dst;
Weight = wgth;
Active = true;

}

}
```

Appendix A2: Source Code of Class Graph:

```
class Graph
{
public int verticesCount;
public int edgesCount;
public int[] remote_Node;
public Edge[] ed;
private static double area = 0.00080642;
private static double p = 0.0000000250188;
public void create_Graph(int vertices,int edges, Edge [] ed1)
{
  this.verticesCount = vertices;
  this.edgesCount = edges;
  this.ed = ed1;}


public double mean_Weight(Edge[] ED)
{
  double mean = 0;
  int n = 0;
  for (int i = 0; i < ED.Length; i++)
  {
    if (ED[i].Active == true)
   {
    mean += ED[i].Weight;
    n++;
   }
  }
  mean = mean / n;
return mean;
}
public double standard_Deviation(Segment[] s, double mean)
{
double standard = 0;
   for (int i = 0; i < s.Length; i++)
   {
    standard += Math.Pow((mean - s[i].nodes.Count), 2);
   }
   standard = (standard) / (s.Length-1);
   standard = Math.Sqrt(standard);
return standard;
}
public double standard_Deviation(Edge[] ED,double mean)
{
  double standard = 0;
  for (int i=0;i<ED.Length;i++)
  {
   standard +=Math.Pow((mean - ED[i].Weight),2);
  }
  standard = (standard)/(ED.Length-1));
  standard = Math.Sqrt(standard);
  return standard;
}
public double coeffcient_Variation(double Std_DV,double mean)
{
double co_V = 0;
co_V = Std_DV / mean;
return co_V;
}
//THIS METHOD IS USED FOR BUS300
public void find_remote_node(string[] lines)
{
this.remote_Node = new int[300];
for (int i=0;i<lines.Length;i++)
{
  int count = 1;
  string remote=null;
  foreach(char c in lines[i])
 {
   if (count <= 4)
  {
   if (c != 32)
    remote += Convert.ToString(c);
  }
  count++;
  }
  remote_Node[i] = Convert.ToInt16(remote);
 }
}
}
```

Appendix A3: Source Codes of Create Graph Manually:

```
private void button1_Click(object sender, EventArgs e)
{
N = Convert.ToInt16(textBox5.Text);
//Create graph's edge object with the number of edge that input in textBox4
if (check == true){

   edg = new Edge[Convert.ToInt16(textBox4.Text)];
   check = false;
   for (int j = 0; j < edg.Length; j++
     edg[j] = new Edge();
}
//Adding edge details....add_Edge(source,Destination,Weight)
if ((i < edg.Length) && (textBox6.Text != ""))
{
edg[i].add_Edge(Convert.ToInt16(textBox1.Text) - 1, Convert.ToInt16(textBox2.Text) - 1,
Convert.ToInt16(textBox3.Text));
i++;}
if (textBox6.Text == "")
{
  MessageBox.Show("Please enter the number of segment!!!");
}
//When i==edge.length inserting is finished
if (i==edg.Length-1)
{
  textBox1.Enabled = false;
  textBox2.Enabled = false;
  textBox3.Enabled = false;
  button1.Enabled = false;
  G.create_Graph(N,Convert.ToInt16(textBox4.Text),edg);

MessageBox.Show("Graph is created\n\n"+"Number of Vertices=\n"+textBox5.Text+"Number of Edges="+textBox4.Text);

}
```

Appendix A4: Source of Method Create-Graph_Text File:

```
public Graph creat_Graph_text(string [] allLines,int edge_length,int vertices)
{
ShowGrapgdetails sh = new ShowGrapgdetails();
sh.Show();
  if (vertices == 300)
  {

string[] lines = File.ReadAllLines("C:\\Users\\Administrator\\Desktop\\TrustNode\\TrustNode\\NewFolder1\\Bus300 remote.txt");

   find_remote_node(lines);
  }
Graph G=new Graph();
G.ed = new Edge[edge_length];
   for (int i = 0; i < edge_length; i++)
   G.ed[i] = new Edge();
```

```
string source = null;
string dest = null;
string weigth = null;
G.verticesCount = vertices;
G.edgesCount = edge_length;
int edgecol = 0;
   for (int i = 0; i < allLines.Length; i++)
   {
   int count = 1;
   source = null;
   dest = null;
   weigth = null;
     foreach (char c in allLines[i])
     {
       if (count <= 4)
      {
         if (c != 32)
            source += Convert.ToString(c);
       }
      if ((count >= 6) && (count <= 9))
     {
         if (c != 32)
            dest += Convert.ToString(c);
     }
      if ((count >= 20) && (count <= 29))
      {
          if (c != 32)
              weigth += Convert.ToString(c);
       }
  count++;
  }
if (vertices == 300)
{
for(int k=0;k<300;k++)
{
if(Convert.ToInt16(source)==remote_Node[k])
G.ed[edgecol].Source = k;
if (Convert.ToInt16(dest) == remote_Node[k])
G.ed[edgecol].Destination= k;
G.ed[edgecol].Active = true;
}
}
else
{
G.ed[edgecol].Source = Convert.ToInt16(source) - 1;


G.ed[edgecol].Destination = Convert.ToInt16(dest) - 1;
G.ed[edgecol].Active = true;
}
if (Convert.ToDouble(weigth) > 0.00000000000000000000)
{
double x = ((Convert.ToDouble(weigth)* area) / p);
x = (x * 3) / 299792.458;
x = Math.Floor(x *1000);
G.ed[edgecol].Weight = Convert.ToInt16(x);

}
else
{
```

```
G.ed[edgecol].Weight = 1;
}
sh.input_text("\t\t\t\t\t" + source + "\t\t\t\t" + dest + "\t\t\t" + G.ed[edgecol].Weight);
edgecol++;

}
return G;
}
```

Appendix A5: Source Code of Class Minimum_Spanning_Tree:

```
class Minimum_Spanning_Tree
{

private Graph Mst = new Graph();


private static int Find(Subset[] subsets, int i)
{
  if (subsets[i].parent != i)
  subsets[i].parent = Find(subsets, subsets[i].parent);

  return subsets[i].parent;
}

public static void exchange(Edge[] data, int m, int n)
{
Edge temporary;

temporary = data[m];
data[m] = data[n];
data[n] = temporary;
}
public static void IntArrayBubbleSort(Edge[] data)
{
int i, j;
int N = data.Length;

for (j = N - 1; j > 0; j--)
{
for (i = 0; i < j; i++)
{
if (data[i].Weight > data[i + 1].Weight)
exchange(data, i, i + 1);
}
}
}

private static void Union(Subset[] subsets, int x, int y)
{
int xroot = Find(subsets, x);
int yroot = Find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
      subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
       subsets[yroot].parent = xroot;
    else
    {
          subsets[yroot].parent = xroot;
         ++subsets[xroot].rank;
     }
   }

//Prepare the string to show the MST details
public  string Print(Edge[] result,int e)
{
string s = null; ;
  for (int i = 0; i < e-1; ++i)
  {
     s += "\t"+(result[i].Source) + "\t\t" + (result[i].Destination) + "\t\t\t" + result[i].Weight;
     s += Environment.NewLine;
   }
return s;
}

//Disjointset method used for segmentation the difference with kruskal is: kruskal return grap
h disjointset return subset
public  Subset[] disjointset(Graph graph)
{

int verticesCount = graph.verticesCount;

//Initial subset and edge array object
Edge[] result = new Edge[verticesCount];
Subset[] subsets = new Subset[verticesCount];
for (int k = 0; k < verticesCount; k++)
{
result[k] = new Edge();
subsets[k] = new Subset();
}
//-----------------------------------------

int i = 0;
int e = 0;

Array.Sort(graph.ed, delegate (Edge a, Edge b)
{
return a.Weight.CompareTo(b.Weight);
});



for (int v = 0; v < graph.verticesCount; ++v)
{
subsets[v].parent = v;
subsets[v].rank = 0;
}

while (i < graph.ed.Length)
{
Edge nextEdge = graph.ed[i++];
int x = Find(subsets, nextEdge.Source);
int y = Find(subsets, nextEdge.Destination);

if (x != y)
{
result[e++] = nextEdge;
Union(subsets, x, y);
}
}



return subsets;

}
public Graph  Mst_Graph(Graph graph)
{
int verticesCount = graph.verticesCount;

//Initial subset and edge array object
Edge[] result = new Edge[verticesCount];
Subset[] subsets = new Subset[verticesCount];
for(int k=0;k<verticesCount;k++)
{
result[k] = new Edge();
subsets[k] = new Subset();
}
//----------------------------------

int i = 0;
int e = 0;

// IntArrayBubbleSort(graph.ed);
//sorting the edge by weight

Array.Sort(graph.ed, delegate (Edge a, Edge b)
{
return a.Weight.CompareTo(b.Weight);
});


//at first each node's parent  is themself and their rank is 0
for (int v = 0; v < verticesCount; ++v)
{
subsets[v].parent = v;
subsets[v].rank = 0;
}

while (e < verticesCount - 1)
{
Edge nextEdge = graph.ed[i++];
int x = Find(subsets, nextEdge.Source);
int y = Find(subsets, nextEdge.Destination);

if (x != y)
{
result[e++] = nextEdge;
Union(subsets, x, y);
}
}

Mst.ed = result;
Mst.verticesCount = verticesCount;
Mst.edgesCount = verticesCount-1;

return Mst;
```

```
}

//Finding minimum degree of each edge & normalizing the edge weight (edge.weight/maximum weight)
public  void MinDegree(Graph MST)
{
int dest;
int src;
int maxWeight = 0;
maxWeight = MST.ed[0].Weight;
for (int i = 0; i < MST.edgesCount; i++)
{
maxWeight = Math.Max(maxWeight, MST.ed[i].Weight);
int scount = 0;
int dcount = 0;
src = MST.ed[i].Source;
dest = MST.ed[i].Destination;
for (int j = 0; j < MST.edgesCount; j++)
{
if ((src == MST.ed[j].Source) || (src == MST.ed[j].Destination))
{
scount++;
}

if ((dest == MST.ed[j].Source) || (dest == MST.ed[j].Destination))
{
dcount++;
}
}
MST.ed[i].minDegree = Math.Min(scount, dcount);
MST.ed[i].maxdeg = Math.Max(scount,dcount);
}

//calculating the normalizeweight
for (int j = 0; j < MST.edgesCount; j++)
{
MST.ed[j].normalizeWeight = Convert.ToDecimal(Convert.ToDecimal(MST.ed[j].Weight) /
  Convert.ToDecimal(maxWeight));

}
}


}
```

Appendix A6: Source Code of Class LPP:

```
class LPP
{
//STEP2
MLApp.MLApp matlab = new MLApp.MLApp();
private double alpha = 1;
private double beta = 0.5;
public double[,] LPP2(int Numberof_Borderin,int Numberof_interlink,Edge[] Intersegment,
Bordering_Node_Details[] B_S)
{
double[,] f = new double[Numberof_Borderin, 1];
double[,] intcon = new double[1, Numberof_Borderin];
```

```
double[,] A = new double[Numberof_interlink, Numberof_Borderin];
double[,] b = new double[Numberof_interlink,1];
double[,] Aeq = new double[1,Numberof_Borderin];
double[,] beq = new double[1,1];
double[,] lb = new double[1,Numberof_Borderin];
double[,] ub = new double[1,Numberof_Borderin];

for (int i = 0; i < Numberof_Borderin; i++)
for (int j = 0; j < Numberof_Borderin; j++)
f[i, 0] = 1;


for (int i = 0; i < Numberof_Borderin; i++)
{

lb[0, i] = 0;
ub[0, i] = 1;
intcon[0, i] = i + 1;
}
//Fill Array "A"
for(int i=0;i<Numberof_interlink;i++)
{
int index1;
int index2;
index1 = Find_for_Array_A(Intersegment[i].Source, B_S);
index2 = Find_for_Array_A(Intersegment[i].Destination, B_S);
A[i,index1] = -1;
A[i, index2] = -1;
b[i, 0] = -1;
}

//Executing matlab function
int[] ans = new int[Numberof_Borderin];
object res = null;
matlab.Feval("intlinprog", 4, out res, f, intcon, A, b, Aeq, beq, lb, ub);

object[] lppresult = res as object[];
double[,] sln = lppresult[0] as double[,];

return sln;
}


public int Find_for_Array_A(int node,Bordering_Node_Details[] B_s)
{

int j = 0;
int ans = 0;
for(int i=0;i<B_s.Length;i++)
{
for (int k=0;k<B_s[i].BNode.Count;k++)
{
if (node != B_s[i].BNode[k])
j++;
else
{
ans = j;

}
}
}

return ans;
}

//This method solve the lpp problem and return N-K number of edges (ESS)
public Edge[] LPP1(Graph Mst, double Num_segMent)
{
double[,] f = new double[Mst.edgesCount, 1];
double[,] intcon = new double[1, Mst.edgesCount];
double[,] A = new double[Mst.edgesCount, Mst.edgesCount];
double[,] b = new double[Mst.edgesCount, 1];
double[,] Aeq = new double[1, Mst.edgesCount];
double[,] beq = new double[1, 1];
double[,] lb = new double[1, Mst.edgesCount];
double[,] ub = new double[1, Mst.edgesCount];

////Preparing Subjects to for mix integer linear programming
for (int i = 0; i < Mst.edgesCount; i++)
for (int j = 0; j < Mst.edgesCount; j++)
A[i, j] = 0;


for (int i = 0; i < Mst.edgesCount; i++)
{
A[i, i] = 1;
lb[0, i] = 0;
ub[0, i] = 1;
intcon[0, i] = i + 1;
b[i, 0] = Mst.ed[i].minDegree;
Aeq[0, i] = 1;
Mst.ed[i].we = (Convert.ToDecimal(alpha) * Convert.ToDecimal(Mst.ed[i].minDegree)) + (
Convert.ToDecimal(beta) * Convert.ToDecimal(Mst.ed[i].normalizeWeight));

}
for (int i = 0; i < Mst.edgesCount; i++)
f[i, 0] = Convert.ToDouble(-Mst.ed[i].we);
beq[0, 0] = Num_segMent - 1;
//-------------------------------------------------------------------


//------------------Executing matlab function
int[] ans = new int[Mst.edgesCount];
object res = null;
matlab.Feval("intlinprog",4, out res, f, intcon, A, b, Aeq, beq, lb, ub);

//--------------------------------------------------

//----------------------------Access to out put of the matlab function------------------------
object[] lppresult = res as object[];
double[,] sln = lppresult[0] as double[,];


//Avtive value changed to false for selected edge from the LPP
for (int i = 0; i < sln.Length; i++)
{
if (sln[i, 0] == 1)
Mst.ed[i].Active = false;

}
```

```
Edge[] rss = new Edge[Mst.verticesCount];
for (int i = 0; i < Mst.verticesCount; i++)
rss[i] = new Edge();
Graph Mstss = new Graph();
Mstss.ed = rss;
Mstss.verticesCount = Mst.verticesCount;
Mstss.edgesCount = Mst.verticesCount - Convert.ToInt16(Num_segMent);
int o = 0;


//preparing Ess edge(matrices)----------------------------------
for (int j = 0; j < Mst.edgesCount; j++)
{
if (Mst.ed[j].Active != false)

{
Mstss.ed[o].Destination = Mst.ed[j].Destination;
Mstss.ed[o].Active = Mst.ed[j].Active;
Mstss.ed[o].Source = Mst.ed[j].Source;
Mstss.ed[o].Weight = Mst.ed[j].Weight;
o++;
}

}
//---------------------------------------------------------------

return Mstss.ed;

}
}
}
```

Appendix A7: Source Code of Class Delta_Min:

```
class Delta_min
{
public Segment delta_computation(Segment S_i, Segment S_j, Graph MST,Graph Mstss)
{
Segment dlt_M = new Segment();
int u = 999;
int counting = 0;
int y = 0;
Edge[] E_Psi = new Edge[S_i.nodes.Count - 1];
Edge[] e_psi_cutted=new Edge[E_Psi.Length-1];

Minimum_Spanning_Tree disj = new Minimum_Spanning_Tree();

Graph cutt = new Graph();

Segment[] segment_Set_U;
Segment s1 = new Segment();
Segment[] delta_min_set;

bool chq = false;
bool chq1 = false;

//Finding U---------------------------------------->Z
for (int i = 0; i < S_i.nodes.Count; i++)
```

```
{
for (int j = 0; j < S_j.nodes.Count; j++)
{
for (int k = 0; k < MST.ed.Length; k++)
{
if (((S_i.nodes[i] == MST.ed[k].Source) || (S_i.nodes[i] == MST.ed[k].Destination)) && ((S_j.nodes[j] == MST.ed[k].Source) || (S_j.nodes[j] == MST.ed[k].Destination)))
{
u = S_i.nodes[i];
}
}
}
}

//-----------------------------------------------------------------
if (u == 999)
{
dlt_M.nodes.Add(-1);
}
else
{
for (int i = 0; i < E_Psi.Length; i++)
E_Psi[i] = new Edge();

int n = 0;
//finding E_Psi-------------------------------------
for (int i = 0; i < Mstss.ed.Length; i++)
{
chq = false;
chq1 = false;
for (int j = 0; j < S_i.nodes.Count; j++)
{
if ((Mstss.ed[i].Source == S_i.nodes[j])&&(Mstss.ed[i].Active==true))
chq = true;

}
for (int j = 0; j < S_i.nodes.Count; j++)
{
if ((Mstss.ed[i].Destination == S_i.nodes[j]) && (Mstss.ed[i].Active == true))

chq1 = true;

}

if ((chq ==true)&&(chq1==true))
{
if ((Mstss.ed[i].Destination == u) || (Mstss.ed[i].Source == u))
{
E_Psi[y].Active = true;
counting++;
}
E_Psi[y].Destination = Mstss.ed[i].Destination;
E_Psi[y].Source = Mstss.ed[i].Source;
E_Psi[y].Active = true;
E_Psi[y].delta_Seg = true;
Mstss.ed[i].delta_Seg = true;
y++;
}
}
for (int i = 0; i < E_Psi.Length - 1; i++)
e_psi_cutted[i] = new Edge();
delta_min_set = new Segment[E_Psi.Length];
for (int i = 0; i < delta_min_set.Length; i++)
delta_min_set[i] = new Segment();

//-----------------------------------End Finding------------------------
for (int i = 0; i < E_Psi.Length; i++)
{
int y1 = 0;
//-------------cutting selected edge belong to u---------------------
if ((E_Psi[i].Source == u) || (E_Psi[i].Destination == u))
{
E_Psi[i].Active = false;
cutt.edgesCount = E_Psi.Length - 1;
for(int k=0;k<E_Psi.Length;k++)
{

if (E_Psi[k].Active == true)
e_psi_cutted[y1++] = E_Psi[k];
}
cutt.verticesCount = MST.verticesCount;
cutt.ed = e_psi_cutted;

//finding delta_u segment after segmentation with updated graph
segment_Set_U = s1.Segmentation(cutt.verticesCount,cutt);
for (int j = 0; j < cutt.verticesCount; j++)
{
for (int k = 0; k < segment_Set_U[j].nodes.Count; k++)
{
if (segment_Set_U[j].nodes[k] == u)
delta_min_set[n] = segment_Set_U[j];
}
}
n++;



//restoring
E_Psi[i].Active = true;
}
}
dlt_M = delta_min_set[0];
for (int i = 1; i < delta_min_set.Length; i++)
{
if (delta_min_set[i].nodes.Count != 0)
{
if(delta_min_set[i].nodes.Count<dlt_M.nodes.Count)
dlt_M = delta_min_set[i];
}
}

}
return dlt_M;


}
}
```

Appendix A8: Source Code of the Button2_Click Event (Implementation of the Local Search and Initial Tree Partitioning Algorithms):

Source Code of the Algorithms 2.7 -2.9:

```
private void button2_Click(object sender, EventArgs e)

{


int phi=1;
if (textBox6.Text == null)

K = Convert.ToInt16(textBox6.Text);
segment_Set = new Segment[K];
for (int i = 0; i < K; i++)
segment_Set[i] = new Segment();
string s = "";
//Minimum spanning tree(MST) using kruskal algorithm
MST =kruskal.Mst_Graph(G);
//minimum degree of each edge and normalized the weight---This method is in minimum spanning tree class
kruskal.MinDegree(MST);
//show the MST graph details in form
ms.Show();

Mean_Weight = G.mean_Weight(G.ed);
St_Deviation = G.standard_Deviation(G.ed, Mean_Weight);




s = kruskal.Print(MST.ed,MST.verticesCount);
ms.input_Mst_Res(s);

//return(N-K) number of edges
Ess = lpp1.LPP1(MST,K);

Mstss.verticesCount = N;
Mstss.edgesCount = N - K;
Mstss.ed = Ess;

//Initial segment sets
segment_Set = s1.Segmentation(K, Mstss);
Ks= Ks = Convert.ToInt16(Math.Ceiling(Convert.ToDecimal(G.verticesCount) / Convert.ToDecimal(K)));


while (phi < 1)
{
int count = 0;
for (int i = 0; i <K; i++)
{
for (int j = 0; j < K; j++)
{
if ((segment_Set[i].nodes.Count > Ks) && (segment_Set[j].nodes.Count < Ks))
{

delta_Min = computation.delta_computation(segment_Set[i], segment_Se
t[j], MST, Mstss);
if (delta_Min.nodes.Count > 0)
{
if (delta_Min.nodes[0] != -1)
{
if (((Math.Abs(Ks - segment_Set[i].nodes.Count)) + (Math.Abs(Ks -
 segment_Set[j].nodes.Count))) >
(Math.Abs(Ks -
 segment_Set[i].nodes.Count + delta_Min.nodes.Count) + (Math.Abs(Ks
- segment_Set[j].nodes.Count - delta_Min.nodes.Count)))
{
//Add
for (int k = 0; k < delta_Min.nodes.Count; k++)
segment_Set[j].nodes.Add(delta_Min.nodes[k]);

//remove
for (int l = 0; l < delta_Min.nodes.Count; l++)
{
for (int k = 0; k < segment_Set[i].nodes.Count; k++)
{
if (segment_Set[i].nodes[k] == delta_Min.nodes[l])
segment_Set[i].nodes.RemoveAt(k);
}
}
count++;
}
}
}
}
}
}
if (count == 0)
phi = 1;
}
```

Appendix A9: Source Code of Class Borderin_Node_Details:

```
class Bordering_Node_Details
{
public IList<int> BNode;
public int segment_Name { get; set; }



public IList<int> trust_node;

public void add_bordering_node(int x,IList<int> B)
{
bool check = false;
int n;
//check for duplicate
for (int i=0;i<B.Count;i++)
{
if(x==B[i])
{
```

```
check = true;
}
}
if (check == false)
BNode.Add(x);
}




}

}
```

Appendix A10: Source Code of Class Trust_Node_Selection:

```
class Trust_Node_Selection
{
private Bordering_Node_Details V_Trust = new Bordering_Node_Details();
private Segment s1 = new Segment();
private LPP lp2 = new LPP();
private Edge[] L_xy ;
public string ANSWER = "Trus Nodes Are:\n\t";
public Bordering_Node_Details[] trusted(Graph G,Segment[] S)
{
//Initializing Bordering node and Intersegment sets
int count_Inter_Link=0;
int x;
int y;
double[,] ans;
Bordering_Node_Details[] B_s = new Bordering_Node_Details[S.Length];
for (int i = 0; i < S.Length; i++)
{
B_s[i] = new Bordering_Node_Details();
B_s[i].BNode = new List <int> ();
B_s[i].trust_node = new List<int>();
B_s[i].segment_Name = i;
}


//------------------------------------------------------------

//-------------------------------------------
//---------------------------------------
for (int i = 0; i < G.edgesCount; i++)
{

x = s1.find_segment_belongtonode(G.ed[i].Source, S);

y = s1.find_segment_belongtonode(G.ed[i].Destination, S);

if (x != y)
{
B_s[x].add_bordering_node(G.ed[i].Source,B_s[x].BNode);
B_s[y].add_bordering_node(G.ed[i].Destination,B_s[y].BNode);
G.ed[i].Intersegment = true;
count_Inter_Link++;

}
}
//Fill the intersegment link array
L_xy = new Edge[count_Inter_Link];
for (int i = 0; i < L_xy.Length; i++)
L_xy[i] = new Edge();
int m = 0;
for (int i = 0; i < G.edgesCount; i++)
{
if (G.ed[i].Intersegment == true)

{
L_xy[m].Source = G.ed[i].Source;
L_xy[m].Destination = G.ed[i].Destination;
m++;
}
}
//----------------------------------------
int count_bordering = 0;
for (int i = 0; i < S.Length; i++)
count_bordering += B_s[i].BNode.Count;

ans=lp2.LPP2(count_bordering,count_Inter_Link,L_xy,B_s);
int count = 0;
//search matlab result in bordering node and convert it to node as string
for(int i=0;i<count_bordering;i++)
{
if (ans[i, 0] != 0)
{
int j = i;
int i1 = 0;

for(int k=0;k<B_s.Length;k++)
{
for(int l=0;l<B_s[k].BNode.Count;l++)
{
if (j == i1)
{
ANSWER += Convert.ToString(B_s[k].BNode[l] + 1) + ",";

B_s[k].trust_node.Add(B_s[k].BNode[l]);
count++;
i1++;
}
else
i1++;

}
}
}
}


ANSWER += Environment.NewLine;
ANSWER += "Number of trust nodes are: " + Convert.ToString(count);

return B_s;
}
}
```

Appendix A11: Source Code of Uniform_Segment:

This class is used to uniform the segments in number of trust systems.

```
class Uniform_Segment
{

public Bordering_Node_Details[] uniform(Bordering_Node_Details[] b_s,Graph G, Segment
[] S, int Number_of_Segment,int num_trustNodes )
{
int balance = 0;
int n ;
int ph = 0;

int temp=0;
int Ns =Convert.ToInt16( Math.Ceiling((Convert.ToDecimal(Number_of_Segment)/Convert.
ToDecimal(num_trustNodes))));
int Ns1= Convert.ToInt16(Math.Ceiling((Convert.ToDecimal(num_trustNodes)/Convert.To
Decimal(Number_of_Segment))));
if (Ns1 > 0)
Ns = Ns1;
while (balance<1)
{
ph = 0;
for(int i=0;i<Number_of_Segment;i++)
{
for(int j=0;j<Number_of_Segment;j++)
{
if((b_s[i].trust_node.Count<Ns)&&(b_s[j].trust_node.Count>Ns))
{
n = 0;
for(int k=0;k<G.ed.Length;k++)
{
//when nodes find to exchange between segments this loop will be finished
if(n<1)
{
bool t1 = false;
bool t2 = false;
bool t3 = false;
//find trust node belongs to oversized bordering node set
for (int h = 0; h < b_s[j].trust_node.Count; h++)
{
if ((b_s[j].trust_node[h] == G.ed[k].Destination) || (b_s[j].trust_node[h] == G.ed[k].Source))
{
temp = b_s[j].trust_node[h];
t1 = true;

}

}
for (int h = 0; h < b_s[i].trust_node.Count; h++)
{
if ((b_s[i].trust_node[h] == G.ed[k].Destination) || (b_s[i].trust_node[h] == G.ed[k].Source))
{
t3 = true;

}

}
if (t1 != t3)
{
int x1 = S[i].find_segment_belongtonode(G.ed[k].Source, S);

int x2 = S[i].find_segment_belongtonode(G.ed[k].Destination, S);
//check other side of the link and be sure that this node belong to undersized segment
if ((((x1 == i) || (x2 == i))) && (x1 != x2))
{
t2 = true;
}

if ((t1 == true) && (t2 == true))
{

n = 1;
//Add
if (temp != G.ed[k].Destination)
b_s[i].trust_node.Add(G.ed[k].Destination);
if (temp != G.ed[k].Source)
b_s[i].trust_node.Add(G.ed[k].Source);

//remove
b_s[j].trust_node.Remove(temp);

ph++;//It checks for balancing

}
}
}
}
}
}
}

if (ph == 0)
balance = 1;
}
return b_s;
}
}
```

Appendix B: IEEE Test System Topologies Databases

Following Tables show the first and last 5 rows of data for IEEE test system topologies. The complete databases are provided in the attached CD.

Appendix B1: IEEE BUS14 Branch Data:

Table 5: IEEE BUS14 branch data.

Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	2	1	1	1	0	0.01938	0.05917	0.0528	0	0	0	0	0	0	0	0	0	0	0	0
2	1	5	1	1	1	0	0.05403	0.22304	0.0492	0	0	0	0	0	0	0	0	0	0	0	0
3	2	3	1	1	1	0	0.04699	0.19797	0.0438	0	0	0	0	0	0	0	0	0	0	0	0
4	2	4	1	1	1	0	0.05811	0.17632	0.034	0	0	0	0	0	0	0	0	0	0	0	0
5	2	5	1	1	1	0	0.05695	0.17388	0.0346	0	0	0	0	0	0	0	0	0	0	0	0
									. . . . .												
16	9	10	1	1	1	0	0.03181	0.0845	0	0	0	0	0	0	0	0	0	0	0	0	0
17	9	14	1	1	1	0	0.12711	0.27038	0	0	0	0	0	0	0	0	0	0	0	0	0
18	10	11	1	1	1	0	0.08205	0.19207	0	0	0	0	0	0	0	0	0	0	0	0	0
19	12	13	1	1	1	0	0.22092	0.19988	0	0	0	0	0	0	0	0	0	0	0	0	0
20	13	14	1	1	1	0	0.17093	0.34802	0	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B2: IEEE BUS30 Branch Data:

Table 6: IEEE BUS30 branch data.

Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	2	1	1	1	0	0.0192	0.0575	0.0528	0	0	0	0	0	0	0	0	0	0	0	0
2	1	3	1	1	1	0	0.0452	0.1652	0.0408	0	0	0	0	0	0	0	0	0	0	0	0
3	2	4	1	1	1	0	0.057	0.1737	0.0368	0	0	0	0	0	0	0	0	0	0	0	0
4	3	4	1	1	1	0	0.0132	0.0379	0.0084	0	0	0	0	0	0	0	0	0	0	0	0
5	2	5	1	1	1	0	0.0472	0.1983	0.0418	0	0	0	0	0	0	0	0	0	0	0	0
									. . . .												
37	27	29	1	1	1	0	0.2198	0.4153	0	0	0	0	0	0	0	0	0	0	0	0	0
38	27	30	1	1	1	0	0.3202	0.6027	0	0	0	0	0	0	0	0	0	0	0	0	0
39	29	30	1	1	1	0	0.2399	0.4533	0	0	0	0	0	0	0	0	0	0	0	0	0
40	8	28	1	1	1	0	0.0636	0.2	0.0428	0	0	0	0	0	0	0	0	0	0	0	0
41	6	28	1	1	1	0	0.0169	0.0599	0.013	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B3: IEEE BUS57 Branch Data:

Table 7: IEEE BUS57 branch data.

Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	2	1	1	1	0	0.0083	0.028	0.129	0	0	0	0	0	0	0	0	0	0	0	0
2	2	3	1	1	1	0	0.0298	0.085	0.0818	0	0	0	0	0	0	0	0	0	0	0	0
3	3	4	1	1	1	0	0.0112	0.0366	0.038	0	0	0	0	0	0	0	0	0	0	0	0
4	4	5	1	1	1	0	0.0625	0.132	0.0258	0	0	0	0	0	0	0	0	0	0	0	0
5	4	6	1	1	1	0	0.043	0.148	0.0348	0	0	0	0	0	0	0	0	0	0	0	0

.
.
.
.
.

76	39	57	1	1	1	0	0	1.355	0	0	0	0	0	0	0.98	0	0	0	0	0	0
77	57	56	1	1	1	0	0.174	0.26	0	0	0	0	0	0	0	0	0	0	0	0	0
78	38	49	1	1	1	0	0.115	0.177	0.003	0	0	0	0	0	0	0	0	0	0	0	0
70	38	48	1	1	1	0	0.0312	0.0482	0	0	0	0	0	0	0	0	0	0	0	0	0
80	9	55	1	1	1	0	0	0.1205	0	0	0	0	0	0	0.94	0	0	0	0	0	0

Appendix B4: IEEE BUS118 Branch Data:

Table 8: IEEE BUS118 branch data.

Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	2	1	1	1	0	0.0303	0.0999	0.0254	0	0	0	0	0	0	0	0	0	0	0	0
2	1	3	1	1	1	0	0.0129	0.0424	0.01082	0	0	0	0	0	0	0	0	0	0	0	0
3	4	5	1	1	1	0	0.00176	0.00798	0.0021	0	0	0	0	0	0	0	0	0	0	0	0
4	3	5	1	1	1	0	0.0241	0.108	0.0284	0	0	0	0	0	0	0	0	0	0	0	0
5	5	6	1	1	1	0	0.0119	0.054	0.01426	0	0	0	0	0	0	0	0	0	0	0	0

.
.
.
.
.
.
.

182	114	115	1	1	1	0	0.0023	0.0104	0.00276	0	0	0	0	0	0	0	0	0	0	0	0
183	68	116	1	1	1	0	0.00034	0.00405	0.164	0	0	0	0	0	0	0	0	0	0	0	0
184	12	117	1	1	1	0	0.0329	0.14	0.0358	0	0	0	0	0	0	0	0	0	0	0	0
185	75	118	1	1	1	0	0.0145	0.0481	0.01198	0	0	0	0	0	0	0	0	0	0	0	0
186	76	118	1	1	1	0	0.0164	0.0544	0.01356	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B5: IEEE BUS300 Branch Data:

Table 9: IEEE BUS300 branch data.

Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	37	9001	1	9	1	2	0.00006	0.00046	0	0	0	75	0	0	1.0082	0	0.9043	1.10435	0.004	0	15	1
2	9001	9005	1	9	1	0	0.0008	0.00348	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	9001	9006	1	9	1	2	0.02439	0.43682	0	0	0	0	9006	0	0.9668	0	0.9391	1.1478	0.00417	0.99	1.01	3
4	9001	9012	1	9	1	2	0.03624	0.64898	0	0	0	0	9012	0	0.9796	0	0.9391	1.1478	0.00417	0.99	1.01	4
5	9005	9051	1	9	1	1	0.01578	0.37486	0	0	0	0	9051	0	1.0435	0	0.9391	1.1478	0.00417	0.99	1.01	5

.

.

.

.

.

407	7039	39	1	1	1	1	0	0.03159	0	0	0	0	0	0	0.965	0	0	0	0	0	0	407
408	7057	57	1	1	1	1	0	0.05347	0	0	0	0	0	0	0.95	0	0	0	0	0	0	408
409	7044	44	1	1	1	1	0	0.18181	0	0	0	0	0	0	0.942	0	0	0	0	0	0	409
410	7055	55	1	1	1	1	0	0.19607	0	0	0	0	0	0	0.942	0	0	0	0	0	0	410
411	7071	71	1	1	1	1	0	0.06896	0	0	0	0	0	0	0.9565	0	0	0	0	0	0	411

Appendix B6: IEEE BUS300 Remote Node Data:

Table 10: IEEE BUS300 remote node data

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	1	0	1.0284	5.95	90	49	0	0	115	0	0	0	0	0	0	1
2	2	1	1	1	0	1.0354	7.74	56	15	0	0	115	0	0	0	0	0	0	2
3	3	1	1	1	0	0.9971	6.64	20	0	0	0	230	0	0	0	0	0	0	3
4	4	1	1	1	0	1.0308	4.71	0	0	0	0	345	0	0	0	0	0	0	4
5	5	1	1	1	0	1.0191	4.68	353	130	0	0	115	0	0	0	0	0	0	5

.

.

.

.

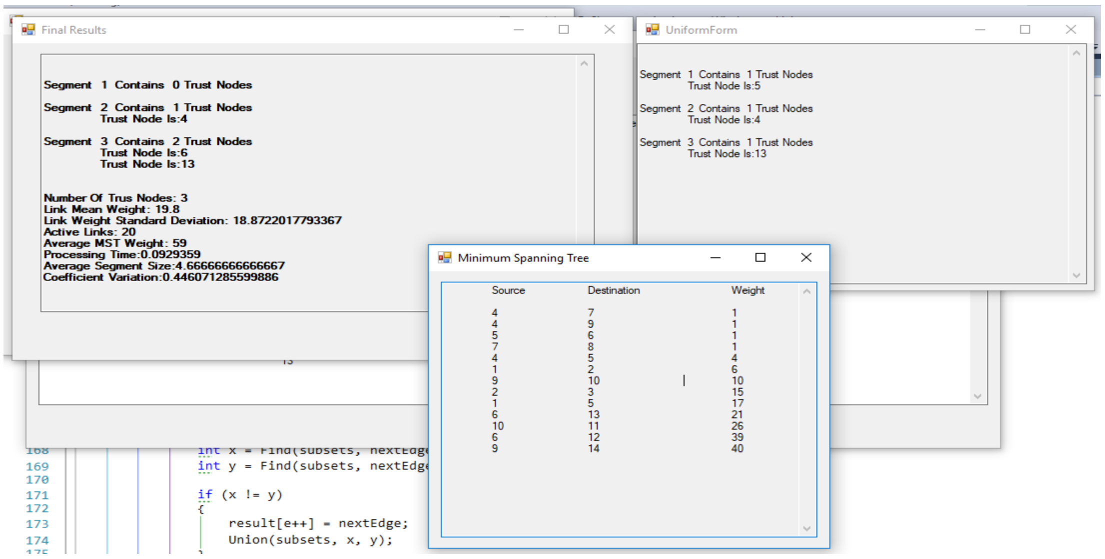
.

296	9055	1	1	9	2	1	-7.54	0	0	8	0	13.8	1	6	-6	0	0	9055	296
297	9071	1	1	9	0	0.9752	-20.48	1.02	0.35	0	0	0.6	0	0	0	0.0005	0	0	297
298	9072	1	1	9	0	0.9803	-19.92	1.02	0.35	0	0	0.6	0	0	0	0.0005	0	0	298
299	9121	1	1	9	0	0.9799	-19.3	3.8	1.25	0	0	6.6	0	0	0	0	0	0	299
300	9533	1	1	9	0	1.0402	-18.24	1.19	0.41	0	0	2.3	0	0	0	0.001	0	0	300

Appendix C: Experimental Results

The following Figures are the screenshots of 5 different databases output results.

Appendix C1: Screenshots of Experimental Results on IEEE Test System of BUS14:



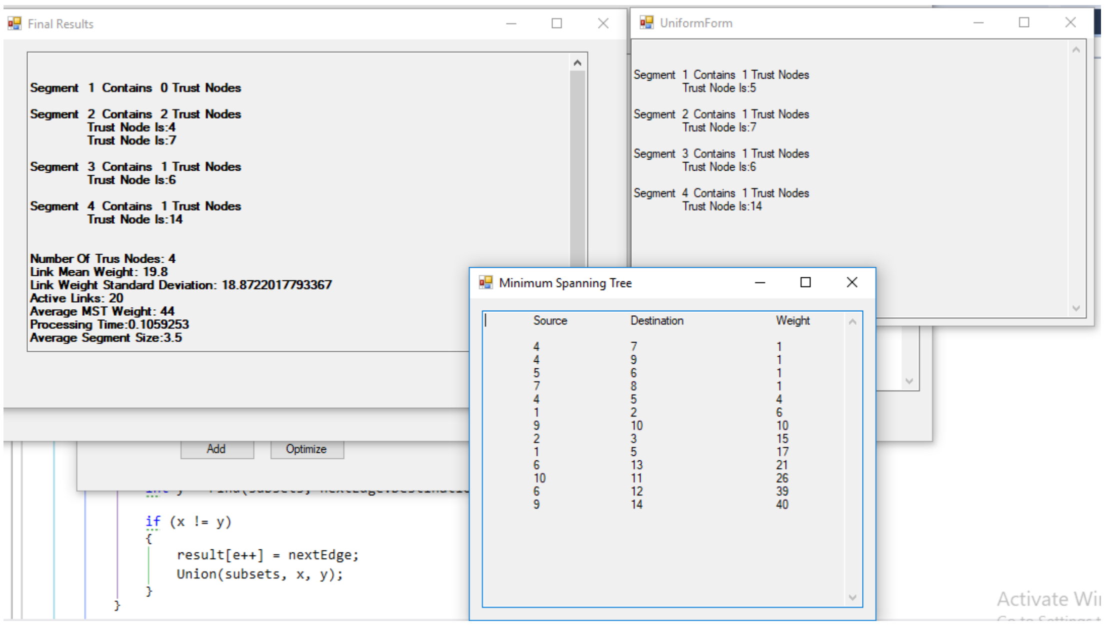
```
Final Results

Segment  1  Contains  0 Trust Nodes

Segment  2  Contains  1 Trust Nodes
         Trust Node Is:4

Segment  3  Contains  2 Trust Nodes
         Trust Node Is:6
         Trust Node Is:13

Number Of Trus Nodes: 3
Link Mean Weight: 19.8
Link Weight Standard Deviation: 18.8722017793367
Active Links: 20
Average MST Weight: 59
Processing Time:0.0929359
Average Segment Size:4.66666666666667
Coefficient Variation:0.446071285599886
```

```
UniformForm

Segment  1  Contains  1 Trust Nodes
         Trust Node Is:5

Segment  2  Contains  1 Trust Nodes
         Trust Node Is:4

Segment  3  Contains  1 Trust Nodes
         Trust Node Is:13
```

```
Minimum Spanning Tree

Source      Destination      Weight

4           7                1
4           9                1
5           6                1
7           8                1
4           5                4
1           2                6
9           10               10
2           3                15
1           5                17
6           13               21
10          11               26
6           12               39
9           14               40
```

```
int x = Find(subsets, nextEdge
int y = Find(subsets, nextEdge

if (x != y)
{
    result[e++] = nextEdge;
    Union(subsets, x, y);
```

Figure 41: The experimental results on IEEE test system of BUS14 with 3 segments.



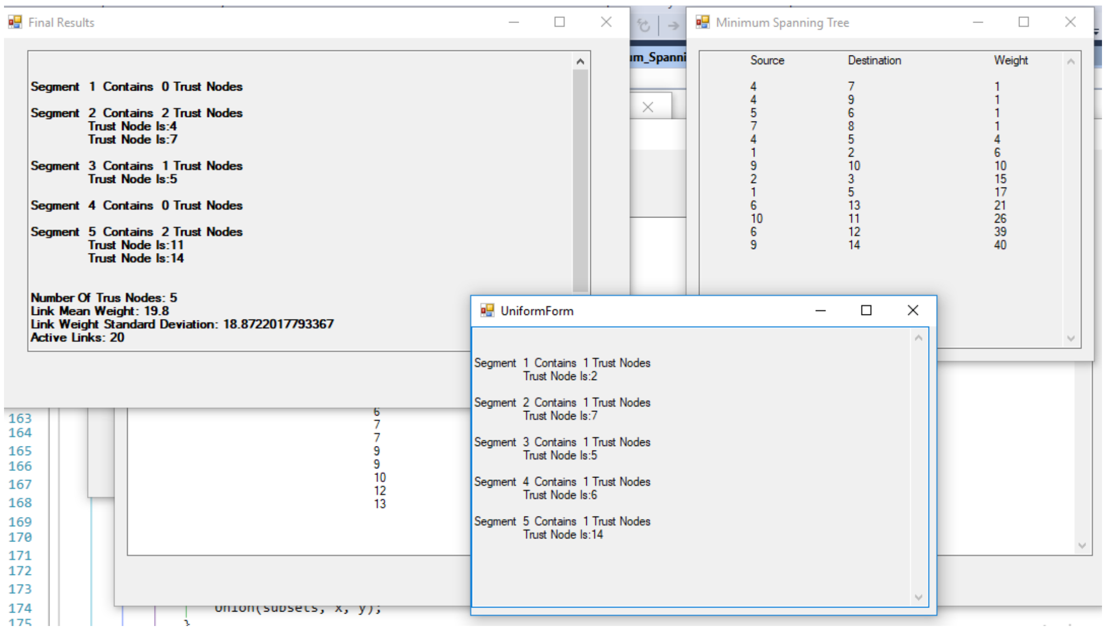
```
Final Results

Segment  1  Contains  0 Trust Nodes

Segment  2  Contains  2 Trust Nodes
         Trust Node Is:4
         Trust Node Is:7

Segment  3  Contains  1 Trust Nodes
         Trust Node Is:6

Segment  4  Contains  1 Trust Nodes
         Trust Node Is:14

Number Of Trus Nodes: 4
Link Mean Weight: 19.8
Link Weight Standard Deviation: 18.8722017793367
Active Links: 20
Average MST Weight: 44
Processing Time:0.1059253
Average Segment Size:3.5
```

```
UniformForm

Segment  1  Contains  1 Trust Nodes
         Trust Node Is:5

Segment  2  Contains  1 Trust Nodes
         Trust Node Is:7

Segment  3  Contains  1 Trust Nodes
         Trust Node Is:6

Segment  4  Contains  1 Trust Nodes
         Trust Node Is:14
```

```
Minimum Spanning Tree

Source      Destination      Weight

4           7                1
4           9                1
5           6                1
7           8                1
4           5                4
1           2                6
9           10               10
2           3                15
1           5                17
6           13               21
10          11               26
6           12               39
9           14               40
```

```
if (x != y)
{
    result[e++] = nextEdge;
    Union(subsets, x, y);
}
}
```

Figure 42: The experimental results on IEEE test system of BUS14 with 4 segments.

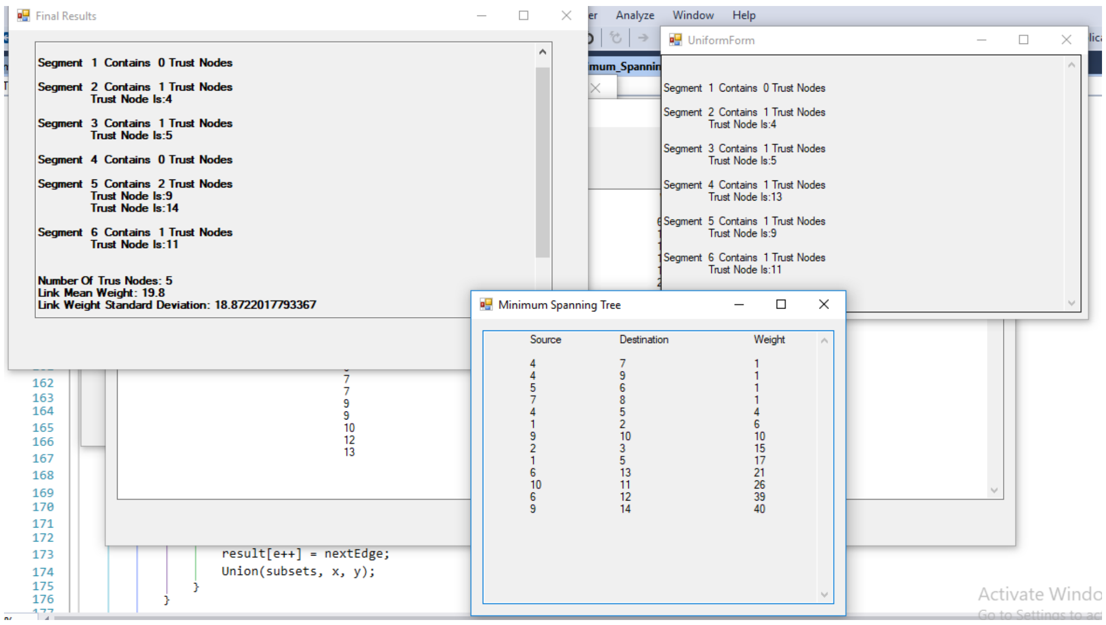


```
Final Results

Segment  1  Contains  0  Trust  Nodes

Segment  2  Contains  2  Trust  Nodes
            Trust  Node  Is:4
            Trust  Node  Is:7

Segment  3  Contains  1  Trust  Nodes
            Trust  Node  Is:5

Segment  4  Contains  0  Trust  Nodes

Segment  5  Contains  2  Trust  Nodes
            Trust  Node  Is:11
            Trust  Node  Is:14

Number Of  Trus Nodes: 5
Link  Mean  Weight:  19.8
Link  Weight  Standard  Deviation:  18.8722017793367
Active  Links:  20
```

```
Minimum Spanning Tree

Source      Destination      Weight
4           7                1
4           9                1
5           6                1
7           8                1
4           5                4
1           2                6
9           10               10
2           3                15
1           5                17
6           13               21
10          11               26
6           12               39
9           14               40
```

```
UniformForm

Segment  1  Contains  1 Trust Nodes
            Trust Node Is:2

Segment  2  Contains  1 Trust Nodes
            Trust Node Is:7

Segment  3  Contains  1 Trust Nodes
            Trust Node Is:5

Segment  4  Contains  1 Trust Nodes
            Trust Node Is:6

Segment  5  Contains  1 Trust Nodes
            Trust Node Is:14
```

Figure 43: The experimental results on IEEE test system of BUS14 with 5 segments.



```
Final Results

Segment  1  Contains  0  Trust  Nodes

Segment  2  Contains  1  Trust  Nodes
            Trust  Node  Is:4

Segment  3  Contains  1  Trust  Nodes
            Trust  Node  Is:5

Segment  4  Contains  0  Trust  Nodes

Segment  5  Contains  2  Trust  Nodes
            Trust  Node  Is:9
            Trust  Node  Is:14

Segment  6  Contains  1  Trust  Nodes
            Trust  Node  Is:11

Number Of  Trus Nodes: 5
Link  Mean  Weight:  19.8
Link  Weight  Standard  Deviation:  18.8722017793367
```

```
UniformForm

Segment  1  Contains  0 Trust Nodes

Segment  2  Contains  1 Trust Nodes
            Trust Node Is:4

Segment  3  Contains  1 Trust Nodes
            Trust Node Is:5

Segment  4  Contains  1 Trust Nodes
            Trust Node Is:13

Segment  5  Contains  1 Trust Nodes
            Trust Node Is:9

Segment  6  Contains  1 Trust Nodes
            Trust Node Is:11
```

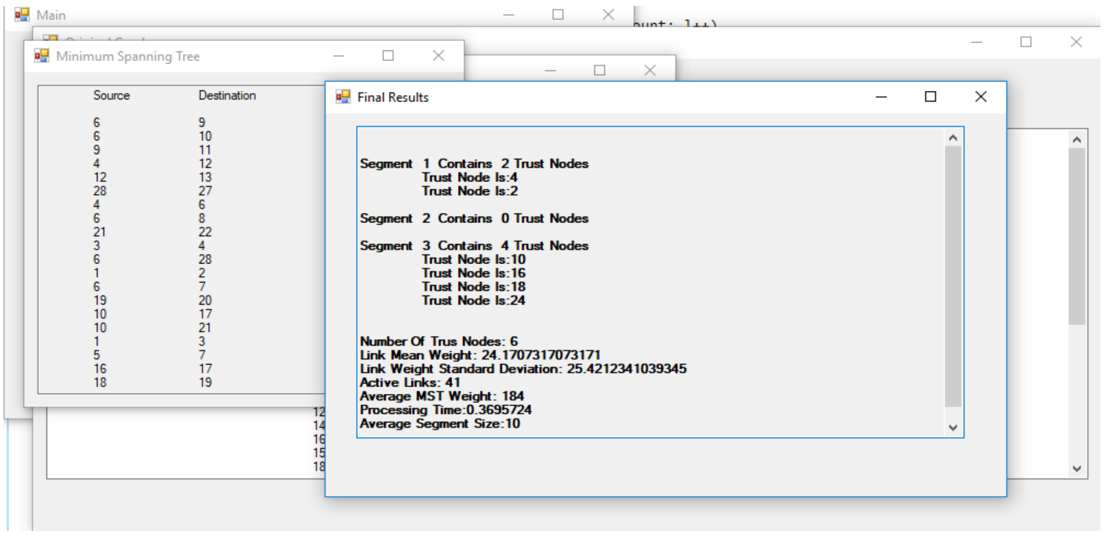
```
Minimum Spanning Tree

Source      Destination      Weight
4           7                1
4           9                1
5           6                1
7           8                1
4           5                4
1           2                6
9           10               10
2           3                15
1           5                17
6           13               21
10          11               26
6           12               39
9           14               40
```

```
result[e++] = nextEdge;
Union(subsets, x, y);
```

Figure 44: The experimental results on IEEE test system of BUS14 with 6 segments.

Appendix C2: Screenshots of Experimental Results on IEEE Test System of BUS30:

Following Figures show the experimental results on IEEE test system of BUS30.



Main

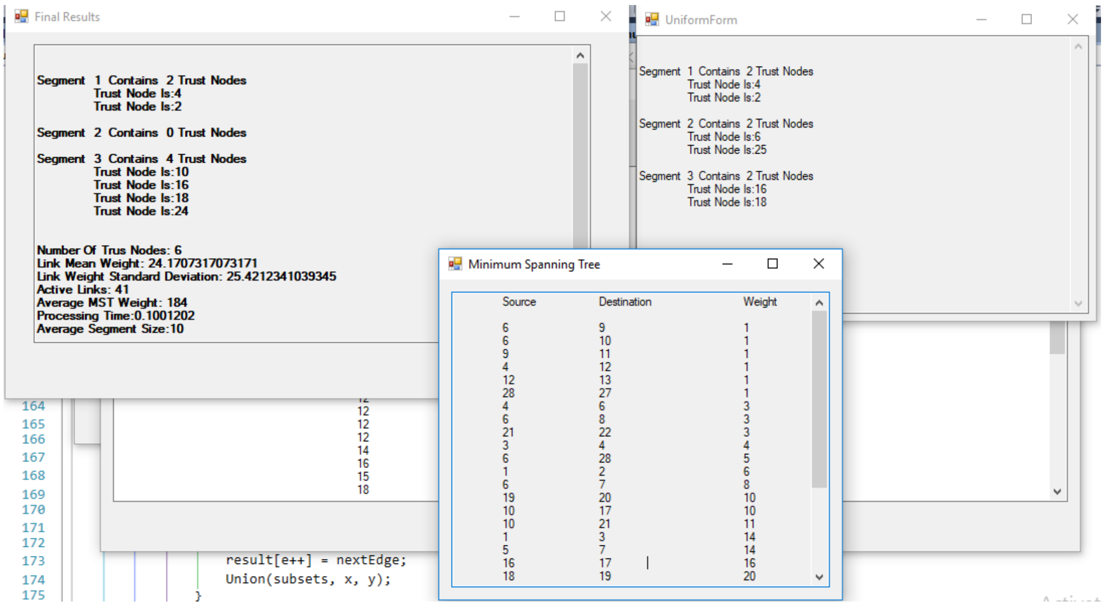
Minimum Spanning Tree

```
Source      Destination

6           9
6           10
9           11
4           12
12          13
28          27
4           6
6           8
21          22
3           4
6           28
1           2
6           7
19          20
10          17
10          21
1           3
5           7
16          17
18          19
```

Final Results

```
Segment  1  Contains  2 Trust Nodes
            Trust Node Is:4
            Trust Node Is:2

Segment  2  Contains  0 Trust Nodes

Segment  3  Contains  4 Trust Nodes
            Trust Node Is:10
            Trust Node Is:16
            Trust Node Is:18
            Trust Node Is:24


Number Of Trus Nodes: 6
Link Mean Weight: 24.1707317073171
Link Weight Standard Deviation: 25.4212341039345
Active Links: 41
Average MST Weight: 184
Processing Time:0.3695724
Average Segment Size:10
```

Figure 45: Experimental parameters for IEEE test system topology BUS30.



Final Results

```
Segment  1  Contains  2 Trust Nodes
            Trust Node Is:4
            Trust Node Is:2

Segment  2  Contains  0 Trust Nodes

Segment  3  Contains  4 Trust Nodes
            Trust Node Is:10
            Trust Node Is:16
            Trust Node Is:18
            Trust Node Is:24


Number Of Trus Nodes: 6
Link Mean Weight: 24.1707317073171
Link Weight Standard Deviation: 25.4212341039345
Active Links: 41
Average MST Weight: 184
Processing Time:0.1001202
Average Segment Size:10
```

UniformForm

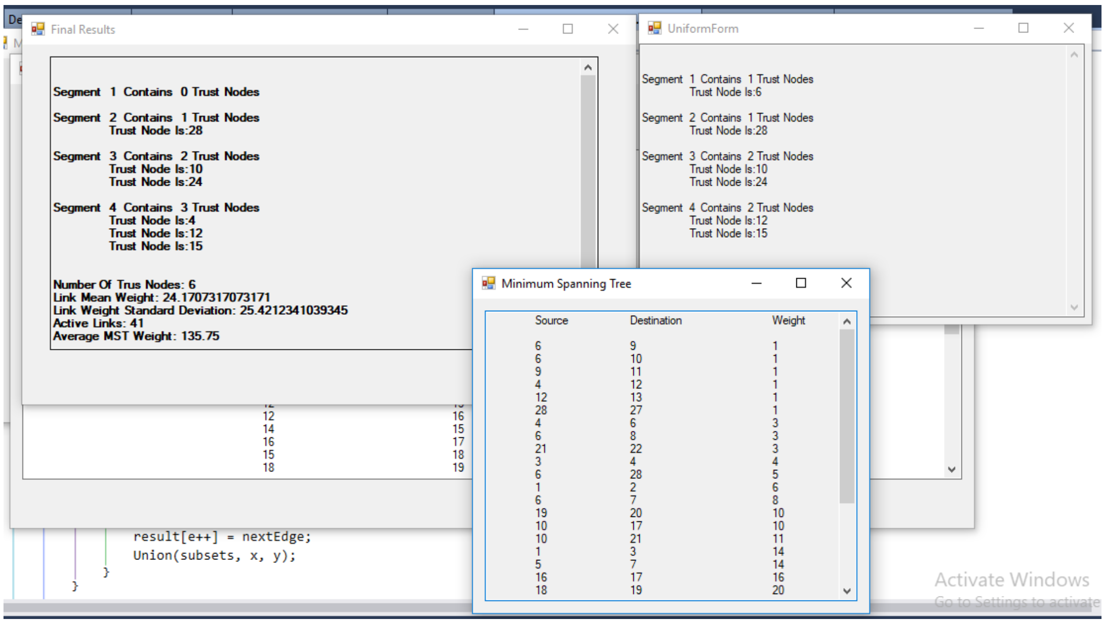
```
Segment  1  Contains  2 Trust Nodes
            Trust Node Is:4
            Trust Node Is:2

Segment  2  Contains  2 Trust Nodes
            Trust Node Is:6
            Trust Node Is:25

Segment  3  Contains  2 Trust Nodes
            Trust Node Is:16
            Trust Node Is:18
```

Minimum Spanning Tree

```
Source      Destination      Weight

6           9                1
6           10               1
9           11               1
4           12               1
12          13               1
28          27               1
4           6                3
6           8                3
21          22               3
3           4                4
6           28               5
1           2                6
6           7                8
19          20               10
10          17               10
10          21               11
1           3                14
5           7                14
16          17               16
18          19               20
```

```
result[e++] = nextEdge;
Union(subsets, x, y);
}
```

Figure 46: The experimental results on IEEE test system of BUS30 with 3 segments.



```
Final Results

Segment  1  Contains  0  Trust  Nodes

Segment  2  Contains  1  Trust  Nodes
         Trust  Node  Is:28

Segment  3  Contains  2  Trust  Nodes
         Trust  Node  Is:10
         Trust  Node  Is:24

Segment  4  Contains  3  Trust  Nodes
         Trust  Node  Is:4
         Trust  Node  Is:12
         Trust  Node  Is:15


Number Of Trus Nodes: 6
Link Mean Weight: 24.1707317073171
Link Weight Standard Deviation: 25.4212341039345
Active Links: 41
Average MST Weight: 135.75
```

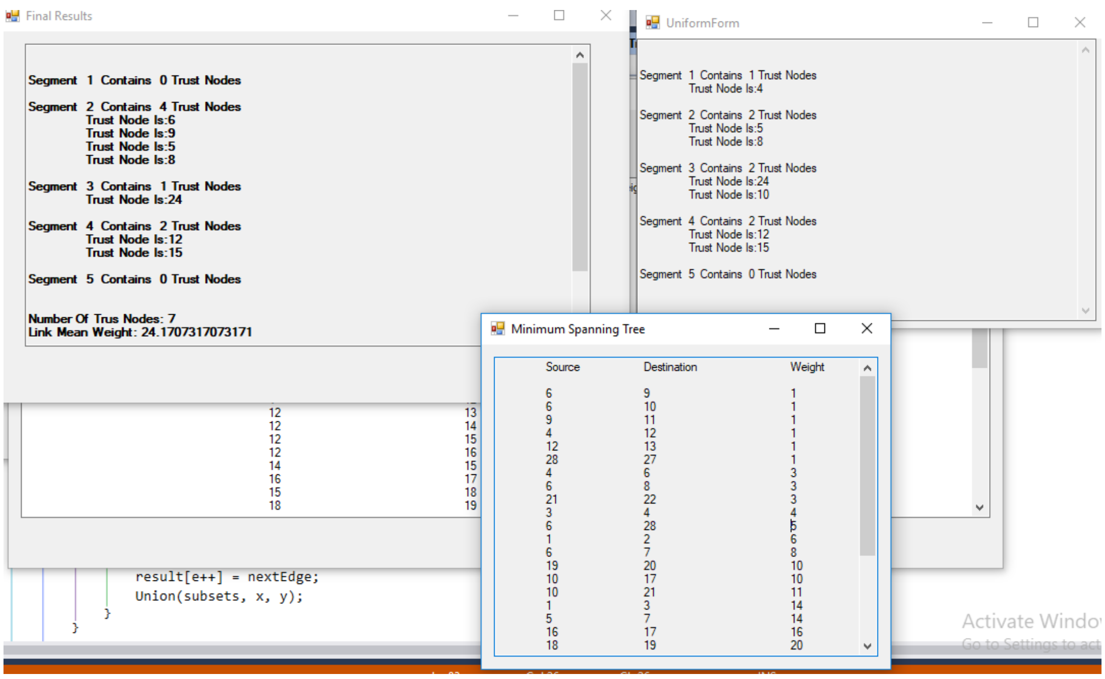
```
UniformForm

Segment  1  Contains  1  Trust Nodes
         Trust Node Is:6

Segment  2  Contains  1  Trust Nodes
         Trust Node Is:28

Segment  3  Contains  2  Trust Nodes
         Trust Node Is:10
         Trust Node Is:24

Segment  4  Contains  2  Trust Nodes
         Trust Node Is:12
         Trust Node Is:15
```

```
Minimum Spanning Tree

Source      Destination      Weight

6           9                1
6           10               1
9           11               1
4           12               1
12          13               1
28          27               1
4           6                3
6           8                3
21          22               3
3           4                4
6           28               5
1           2                6
6           7                8
19          20               10
10          17               10
10          21               11
1           3                14
5           7                14
16          17               16
18          19               20
```

```
12          16
14          15
16          17
15          18
18          19
```

```
                result[e++] = nextEdge;
                Union(subsets, x, y);
            }
        }
```

Figure 47: The experimental results on IEEE test system of BUS30 with 4 segments.



```
Final Results

Segment  1  Contains  0  Trust  Nodes

Segment  2  Contains  4  Trust  Nodes
         Trust  Node  Is:6
         Trust  Node  Is:9
         Trust  Node  Is:5
         Trust  Node  Is:8

Segment  3  Contains  1  Trust  Nodes
         Trust  Node  Is:24

Segment  4  Contains  2  Trust  Nodes
         Trust  Node  Is:12
         Trust  Node  Is:15

Segment  5  Contains  0  Trust  Nodes


Number Of Trus Nodes: 7
Link Mean Weight: 24.1707317073171
```

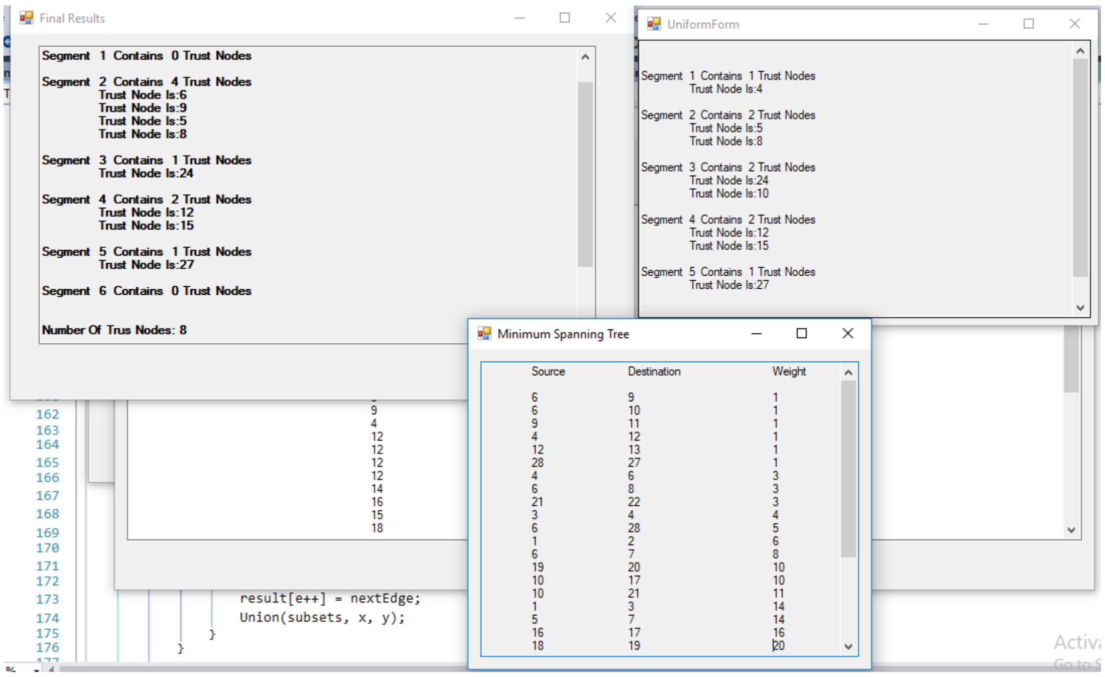
```
UniformForm

Segment  1  Contains  1  Trust Nodes
         Trust Node Is:4

Segment  2  Contains  2  Trust Nodes
         Trust Node Is:5
         Trust Node Is:8

Segment  3  Contains  2  Trust Nodes
         Trust Node Is:24
         Trust Node Is:10

Segment  4  Contains  2  Trust Nodes
         Trust Node Is:12
         Trust Node Is:15

Segment  5  Contains  0  Trust Nodes
```

```
Minimum Spanning Tree

Source      Destination      Weight

6           9                1
6           10               1
9           11               1
4           12               1
12          13               1
28          27               1
4           6                3
6           8                3
21          22               3
3           4                4
6           28               5
1           2                6
6           7                8
19          20               10
10          17               10
10          21               11
1           3                14
5           7                14
16          17               16
18          19               20
```

```
12          13
12          14
12          15
12          16
14          15
16          17
15          18
18          19
```

```
                result[e++] = nextEdge;
                Union(subsets, x, y);
            }
        }
```

Figure 48: The experimental results on IEEE test system of BUS30 with 5 segments.



```
Final Results

Segment  1  Contains  0  Trust  Nodes

Segment  2  Contains  4  Trust  Nodes
         Trust  Node  Is:6
         Trust  Node  Is:9
         Trust  Node  Is:5
         Trust  Node  Is:8

Segment  3  Contains  1  Trust  Nodes
         Trust  Node  Is:24

Segment  4  Contains  2  Trust  Nodes
         Trust  Node  Is:12
         Trust  Node  Is:15

Segment  5  Contains  1  Trust  Nodes
         Trust  Node  Is:27

Segment  6  Contains  0  Trust  Nodes

Number Of  Trus  Nodes: 8
```

```
UniformForm

Segment  1  Contains  1 Trust Nodes
            Trust Node Is:4

Segment  2  Contains  2 Trust Nodes
            Trust Node Is:5
            Trust Node Is:8

Segment  3  Contains  2 Trust Nodes
            Trust Node Is:24
            Trust Node Is:10

Segment  4  Contains  2 Trust Nodes
            Trust Node Is:12
            Trust Node Is:15

Segment  5  Contains  1 Trust Nodes
            Trust Node Is:27
```

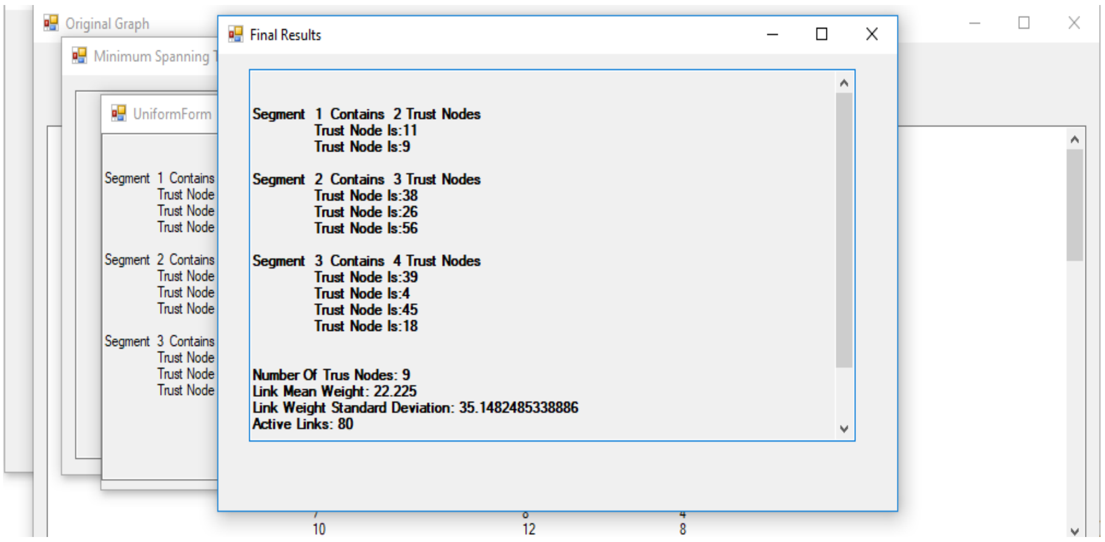
```
Minimum Spanning Tree

        Source        Destination        Weight

        6             9                  1
        6             10                 1
        9             11                 1
        4             12                 1
        12            13                 1
        28            27                 1
        4             6                  3
        6             8                  3
        21            22                 3
        3             4                  4
        6             28                 5
        1             2                  6
        6             7                  8
        19            20                 10
        10            17                 10
        10            21                 11
        1             3                  14
        5             7                  14
        16            17                 16
        18            19                 20
```

```
            result[e++] = nextEdge;
            Union(subsets, x, y);
        }
    }
```

Figure 49: The experimental results on IEEE test system of BUS30 with 6 segments.

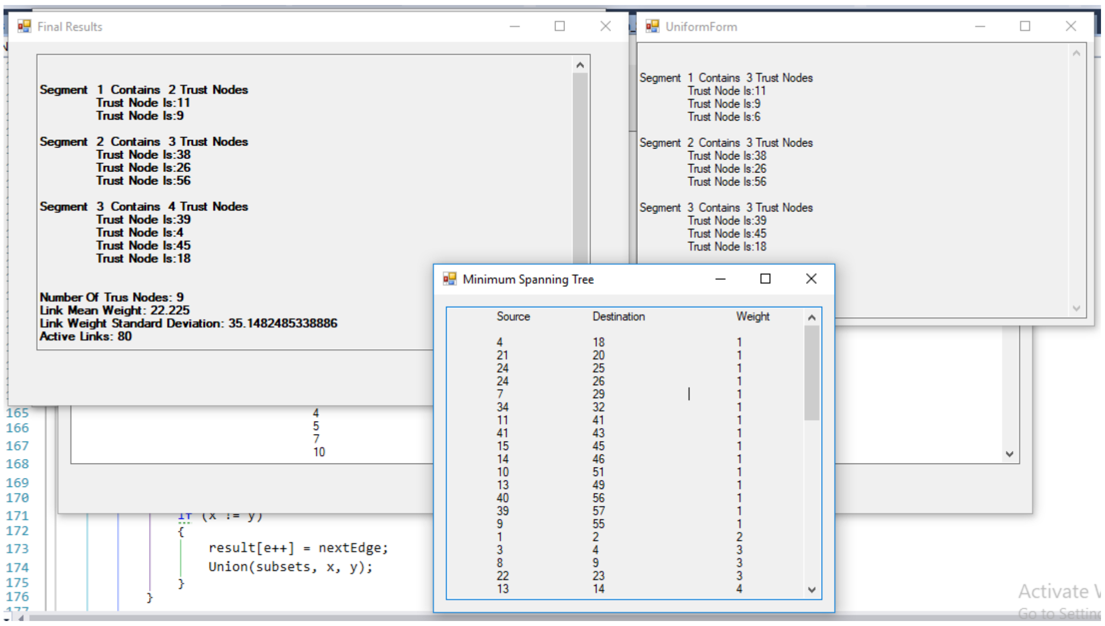
Appendix C3: Experimental Parameters for IEEE Test System Topology BUS57:

Following Figures show the experimental results on IEEE test system of BUS57.



```
Final Results

Segment  1  Contains  2 Trust Nodes
         Trust Node Is:11
         Trust Node Is:9

Segment  2  Contains  3 Trust Nodes
         Trust Node Is:38
         Trust Node Is:26
         Trust Node Is:56

Segment  3  Contains  4 Trust Nodes
         Trust Node Is:39
         Trust Node Is:4
         Trust Node Is:45
         Trust Node Is:18


Number Of Trus Nodes: 9
Link Mean Weight: 22.225
Link Weight Standard Deviation: 35.1482485338886
Active Links: 80
```

Figure 50: Experimental parameters for IEEE test system topology BUS57.



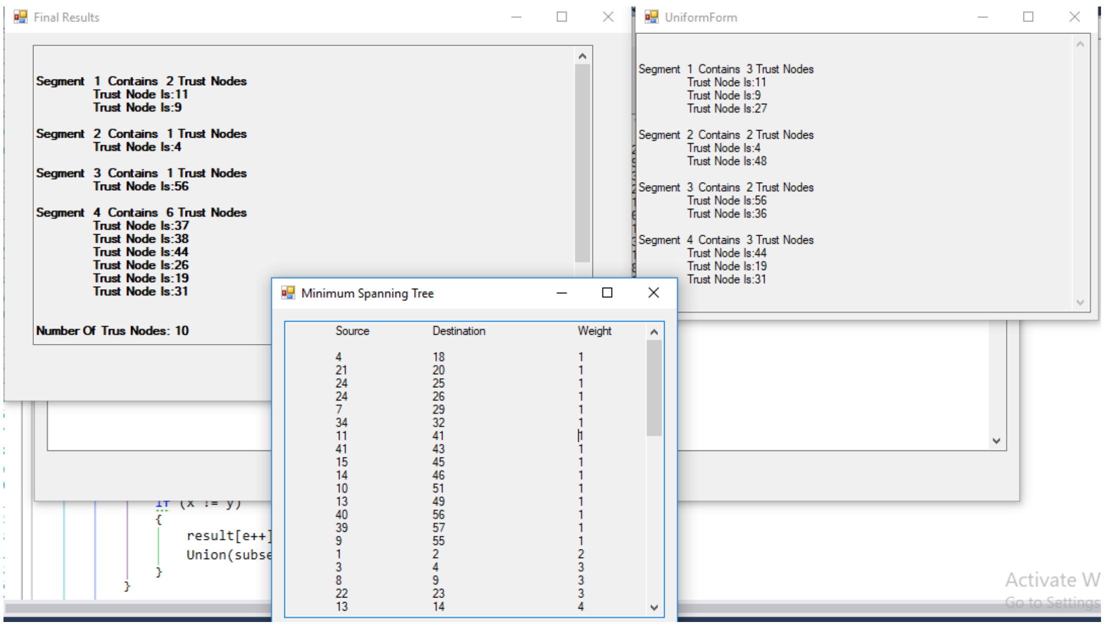
```
Final Results

Segment  1  Contains  2 Trust Nodes
            Trust Node Is:11
            Trust Node Is:9

Segment  2  Contains  3 Trust Nodes
            Trust Node Is:38
            Trust Node Is:26
            Trust Node Is:56

Segment  3  Contains  4 Trust Nodes
            Trust Node Is:39
            Trust Node Is:4
            Trust Node Is:45
            Trust Node Is:18

Number Of Trus Nodes: 9
Link Mean Weight: 22.225
Link Weight Standard Deviation: 35.1482485338886
Active Links: 80
```

```
UniformForm

Segment  1  Contains  3 Trust Nodes
            Trust Node Is:11
            Trust Node Is:9
            Trust Node Is:6

Segment  2  Contains  3 Trust Nodes
            Trust Node Is:38
            Trust Node Is:26
            Trust Node Is:56

Segment  3  Contains  3 Trust Nodes
            Trust Node Is:39
            Trust Node Is:45
            Trust Node Is:18
```

```
Minimum Spanning Tree

      Source        Destination         Weight

      4             18                  1
      21            20                  1
      24            25                  1
      24            26                  1
      7             29                  1
      34            32                  1
      11            41                  1
      41            43                  1
      15            45                  1
      14            46                  1
      10            51                  1
      13            49                  1
      40            56                  1
      39            57                  1
      9             55                  1
      1             2                   2
      3             4                   3
      8             9                   3
      22            23                  3
      13            14                  4
```

```
                if (x != y)
                {
                    result[e++] = nextEdge;
                    Union(subsets, x, y);
                }
            }
```

Figure 51: The experimental results on IEEE test system of BUS57 with 3 segments.

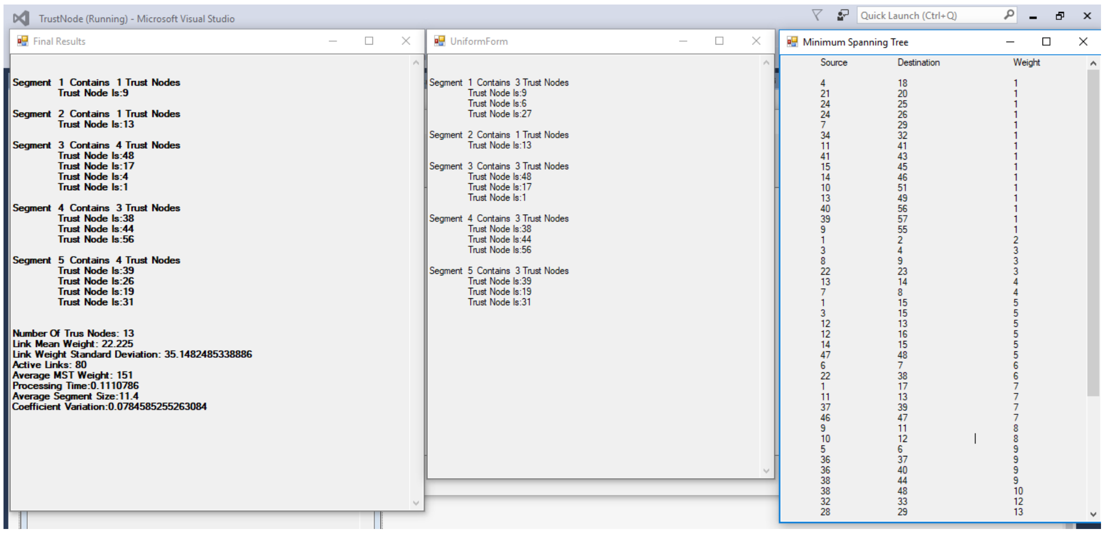


```
Final Results

Segment  1  Contains  2 Trust Nodes
            Trust Node Is:11
            Trust Node Is:9

Segment  2  Contains  1 Trust Nodes
            Trust Node Is:4

Segment  3  Contains  1 Trust Nodes
            Trust Node Is:56

Segment  4  Contains  6 Trust Nodes
            Trust Node Is:37
            Trust Node Is:38
            Trust Node Is:44
            Trust Node Is:26
            Trust Node Is:19
            Trust Node Is:31

Number Of Trus Nodes: 10
```

```
UniformForm

Segment  1  Contains  3 Trust Nodes
            Trust Node Is:11
            Trust Node Is:9
            Trust Node Is:27

Segment  2  Contains  2 Trust Nodes
            Trust Node Is:4
            Trust Node Is:48

Segment  3  Contains  2 Trust Nodes
            Trust Node Is:56
            Trust Node Is:36

Segment  4  Contains  3 Trust Nodes
            Trust Node Is:44
            Trust Node Is:19
            Trust Node Is:31
```

```
Minimum Spanning Tree

      Source        Destination         Weight

      4             18                  1
      21            20                  1
      24            25                  1
      24            26                  1
      7             29                  1
      34            32                  1
      11            41                  1
      41            43                  1
      15            45                  1
      14            46                  1
      10            51                  1
      13            49                  1
      40            56                  1
      39            57                  1
      9             55                  1
      1             2                   2
      3             4                   3
      8             9                   3
      22            23                  3
      13            14                  4
```

Figure 52: The experimental results on IEEE test system of BUS57 with 4 segments.

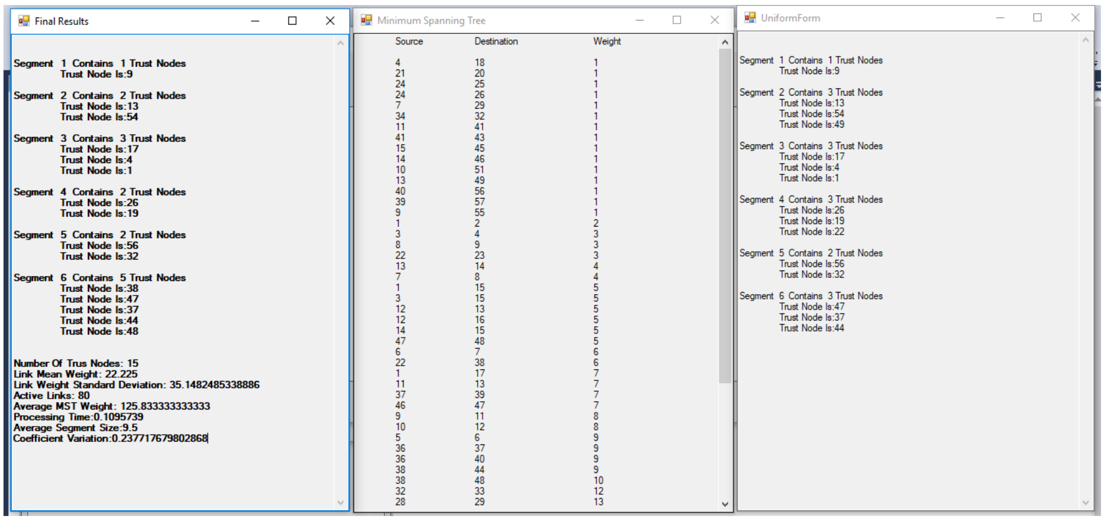


```
Final Results

Segment  1  Contains  1 Trust Nodes
            Trust Node Is:9

Segment  2  Contains  1 Trust Nodes
            Trust Node Is:13

Segment  3  Contains  4 Trust Nodes
            Trust Node Is:48
            Trust Node Is:17
            Trust Node Is:4
            Trust Node Is:1

Segment  4  Contains  3 Trust Nodes
            Trust Node Is:38
            Trust Node Is:44
            Trust Node Is:56

Segment  5  Contains  4 Trust Nodes
            Trust Node Is:39
            Trust Node Is:26
            Trust Node Is:19
            Trust Node Is:31


Number Of Trus Nodes: 13
Link Mean Weight: 22.225
Link Weight Standard Deviation: 35.1482485338886
Active Links: 80
Average MST Weight: 151
Processing Time:0.1110786
Average Segment Size:11.4
Coefficient Variation:0.0784585255263084
```

```
UniformForm

Segment  1  Contains  3 Trust Nodes
            Trust Node Is:9
            Trust Node Is:6
            Trust Node Is:27

Segment  2  Contains  1 Trust Nodes
            Trust Node Is:13

Segment  3  Contains  3 Trust Nodes
            Trust Node Is:48
            Trust Node Is:17
            Trust Node Is:1

Segment  4  Contains  3 Trust Nodes
            Trust Node Is:38
            Trust Node Is:44
            Trust Node Is:56

Segment  5  Contains  3 Trust Nodes
            Trust Node Is:39
            Trust Node Is:19
            Trust Node Is:31
```

```
Minimum Spanning Tree
    Source      Destination      Weight

    4           18               1
    21          20               1
    24          25               1
    24          26               1
    7           29               1
    34          32               1
    11          41               1
    41          43               1
    15          45               1
    14          46               1
    10          51               1
    13          49               1
    40          56               1
    39          57               1
    9           55               1
    1           2                2
    3           4                3
    8           9                3
    22          23               3
    13          14               4
    7           8                4
    1           15               5
    3           15               5
    12          13               5
    12          16               5
    14          15               5
    47          48               5
    6           7                6
    22          38               6
    1           17               7
    11          13               7
    37          39               7
    46          47               7
    9           11               8
    10          12               8
    5           6                9
    36          37               9
    36          40               9
    38          44               9
    38          48               10
    32          33               12
    28          29               13
```

Figure 53: The experimental results on IEEE test system of BUS57 with 5 segments.



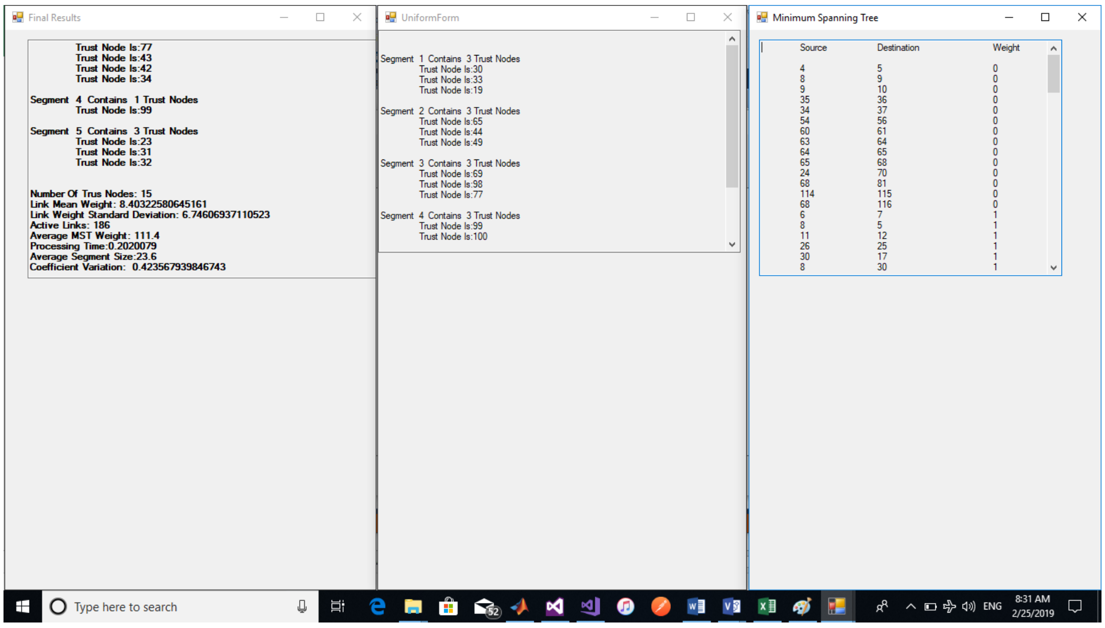
```
Final Results

Segment  1  Contains  1 Trust Nodes
            Trust Node Is:9

Segment  2  Contains  2 Trust Nodes
            Trust Node Is:13
            Trust Node Is:54

Segment  3  Contains  3 Trust Nodes
            Trust Node Is:17
            Trust Node Is:4
            Trust Node Is:1

Segment  4  Contains  2 Trust Nodes
            Trust Node Is:26
            Trust Node Is:19

Segment  5  Contains  2 Trust Nodes
            Trust Node Is:56
            Trust Node Is:32

Segment  6  Contains  5 Trust Nodes
            Trust Node Is:38
            Trust Node Is:47
            Trust Node Is:37
            Trust Node Is:44
            Trust Node Is:48


Number Of Trus Nodes: 15
Link Mean Weight: 22.225
Link Weight Standard Deviation: 35.1482485338886
Active Links: 80
Average MST Weight: 125.833333333333
Processing Time:0.1095739
Average Segment Size:9.5
Coefficient Variation:0.237717679802868
```

```
Minimum Spanning Tree
    Source      Destination      Weight

    4           18               1
    21          20               1
    24          25               1
    24          26               1
    7           29               1
    34          32               1
    11          41               1
    41          43               1
    15          45               1
    14          46               1
    10          51               1
    13          49               1
    40          56               1
    39          57               1
    9           55               1
    1           2                2
    3           4                3
    8           9                3
    22          23               3
    13          14               4
    7           8                4
    1           15               5
    3           15               5
    12          13               5
    12          16               5
    14          15               5
    47          48               5
    6           7                6
    22          38               6
    1           17               7
    11          13               7
    37          39               7
    46          47               7
    9           11               8
    10          12               8
    5           6                9
    36          37               9
    36          40               9
    38          44               9
    38          48               10
    32          33               12
    28          29               13
```

```
UniformForm

Segment  1  Contains  1 Trust Nodes
            Trust Node Is:9

Segment  2  Contains  3 Trust Nodes
            Trust Node Is:13
            Trust Node Is:54
            Trust Node Is:49

Segment  3  Contains  3 Trust Nodes
            Trust Node Is:17
            Trust Node Is:4
            Trust Node Is:1

Segment  4  Contains  3 Trust Nodes
            Trust Node Is:26
            Trust Node Is:19
            Trust Node Is:22

Segment  5  Contains  2 Trust Nodes
            Trust Node Is:56
            Trust Node Is:32

Segment  6  Contains  3 Trust Nodes
            Trust Node Is:47
            Trust Node Is:37
            Trust Node Is:44
```

Figure 54: The experimental results on IEEE test system of BUS57 with 6 segments.

Appendix C4: The Experimental Results on IEEE Test System of BUS118:

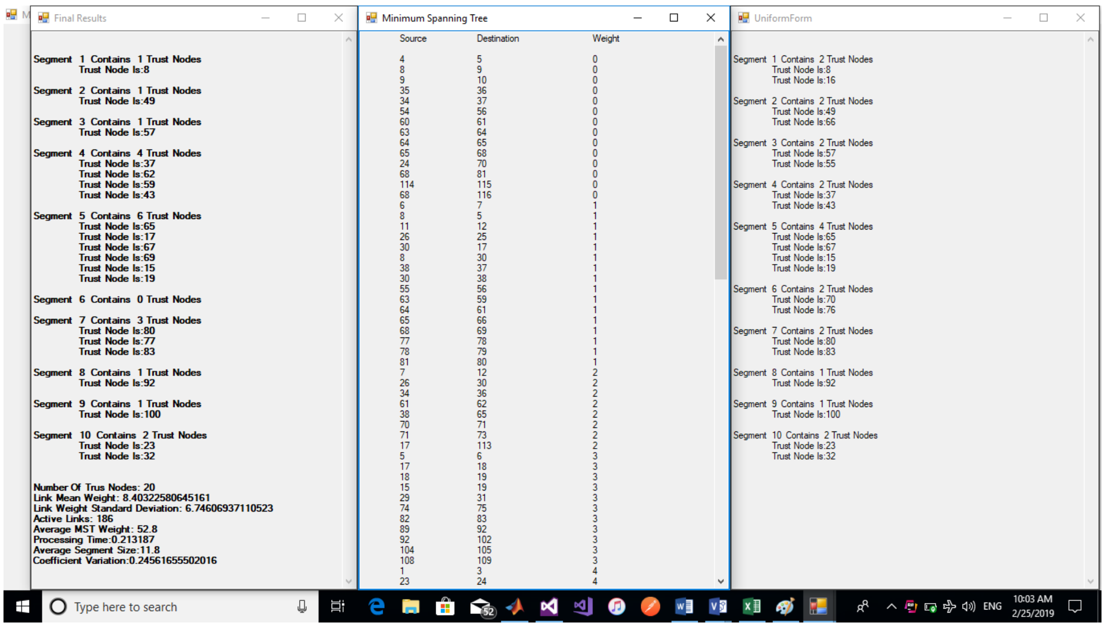


```
                  Trust Node Is:77
                  Trust Node Is:43
                  Trust Node Is:42
                  Trust Node Is:34

Segment  4  Contains  1 Trust Nodes
                  Trust Node Is:99

Segment  5  Contains  3 Trust Nodes
                  Trust Node Is:23
                  Trust Node Is:31
                  Trust Node Is:32


Number Of Trus Nodes: 15
Link Mean Weight: 8.40322580645161
Link Weight Standard Deviation: 6.74606937110523
Active Links: 186
Average MST Weight: 111.4
Processing Time:0.2020079
Average Segment Size:23.6
Coefficient Variation:  0.423567939846743
```

```
Segment  1  Contains  3 Trust Nodes
                Trust Node Is:30
                Trust Node Is:33
                Trust Node Is:19

Segment  2  Contains  3 Trust Nodes
                Trust Node Is:65
                Trust Node Is:44
                Trust Node Is:49

Segment  3  Contains  3 Trust Nodes
                Trust Node Is:69
                Trust Node Is:98
                Trust Node Is:77

Segment  4  Contains  3 Trust Nodes
                Trust Node Is:99
                Trust Node Is:100
```

```
        Source      Destination     Weight

        4           5               0
        8           9               0
        9           10              0
        35          36              0
        34          37              0
        54          56              0
        60          61              0
        63          64              0
        64          65              0
        65          68              0
        24          70              0
        68          81              0
        114         115             0
        68          116             0
        6           7               1
        8           5               1
        11          12              1
        26          25              1
        30          17              1
        8           30              1
```

Figure 55: The experimental results on IEEE test system of BUS118 with 5 segments.

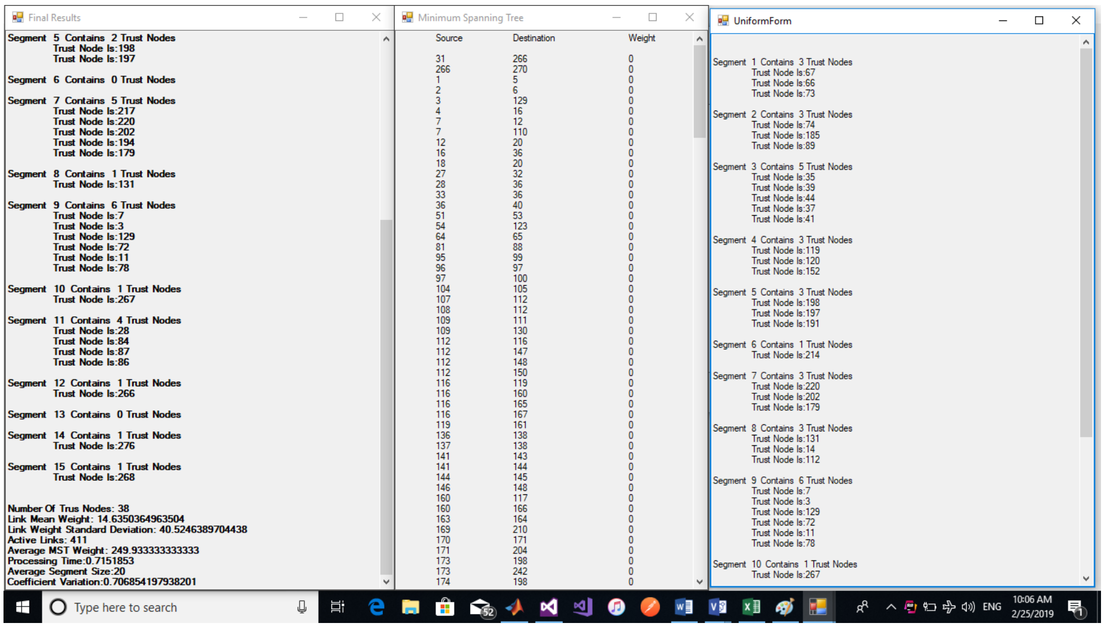


```
Segment  1  Contains  1 Trust Nodes
                 Trust Node Is:8

Segment  2  Contains  1 Trust Nodes
                 Trust Node Is:49

Segment  3  Contains  1 Trust Nodes
                 Trust Node Is:57

Segment  4  Contains  4 Trust Nodes
                 Trust Node Is:37
                 Trust Node Is:62
                 Trust Node Is:59
                 Trust Node Is:43

Segment  5  Contains  6 Trust Nodes
                 Trust Node Is:65
                 Trust Node Is:17
                 Trust Node Is:67
                 Trust Node Is:69
                 Trust Node Is:15
                 Trust Node Is:19

Segment  6  Contains  0 Trust Nodes

Segment  7  Contains  3 Trust Nodes
                 Trust Node Is:80
                 Trust Node Is:77
                 Trust Node Is:83

Segment  8  Contains  1 Trust Nodes
                 Trust Node Is:92

Segment  9  Contains  1 Trust Nodes
                 Trust Node Is:100

Segment  10  Contains  2 Trust Nodes
                 Trust Node Is:23
                 Trust Node Is:32


Number Of Trus Nodes: 20
Link Mean Weight: 8.40322580645161
Link Weight Standard Deviation: 6.74606937110523
Active Links: 186
Average MST Weight: 52.8
Processing Time:0.213187
Average Segment Size:11.8
Coefficient Variation:0.24561655502016
```

```
        Source      Destination     Weight

        4           5               0
        8           9               0
        9           10              0
        35          36              0
        34          37              0
        54          56              0
        60          61              0
        63          64              0
        64          65              0
        65          68              0
        24          70              0
        68          81              0
        114         115             0
        68          116             0
        6           7               1
        8           5               1
        11          12              1
        26          25              1
        30          17              1
        8           30              1
        38          37              1
        30          38              1
        55          56              1
        63          59              1
        64          61              1
        65          66              1
        68          69              1
        77          78              1
        78          79              1
        81          80              1
        7           12              2
        26          30              2
        34          36              2
        61          62              2
        38          65              2
        70          71              2
        71          73              2
        17          113             2
        5           6               3
        17          18              3
        18          19              3
        15          19              3
        29          31              3
        74          75              3
        82          83              3
        89          92              3
        92          102             3
        104         105             3
        108         109             3
        1           3               4
        23          24              4
```

```
Segment  1  Contains  2 Trust Nodes
               Trust Node Is:8
               Trust Node Is:16

Segment  2  Contains  2 Trust Nodes
               Trust Node Is:49
               Trust Node Is:66

Segment  3  Contains  2 Trust Nodes
               Trust Node Is:57
               Trust Node Is:55

Segment  4  Contains  2 Trust Nodes
               Trust Node Is:37
               Trust Node Is:43

Segment  5  Contains  4 Trust Nodes
               Trust Node Is:65
               Trust Node Is:67
               Trust Node Is:15
               Trust Node Is:19

Segment  6  Contains  2 Trust Nodes
               Trust Node Is:70
               Trust Node Is:76

Segment  7  Contains  2 Trust Nodes
               Trust Node Is:80
               Trust Node Is:83

Segment  8  Contains  1 Trust Nodes
               Trust Node Is:92

Segment  9  Contains  1 Trust Nodes
               Trust Node Is:100

Segment  10  Contains  2 Trust Nodes
               Trust Node Is:23
               Trust Node Is:32
```

Figure 56: The experimental results on IEEE test system of BUS118 with 10 segments.

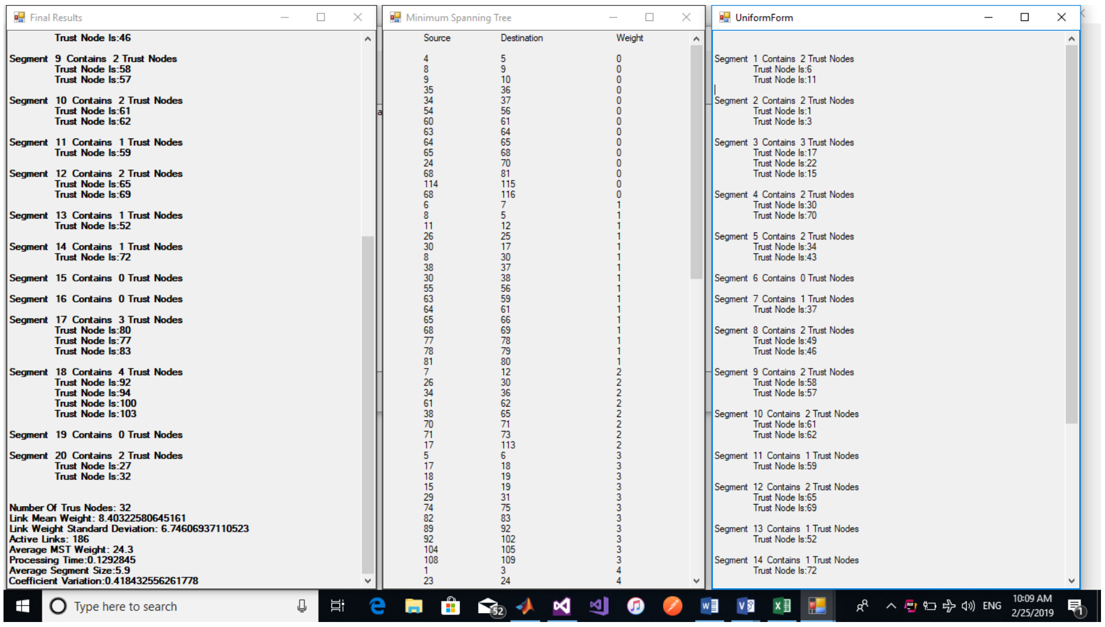


```
Segment  5  Contains  2 Trust Nodes
            Trust Node Is:198
            Trust Node Is:197

Segment  6  Contains  0 Trust Nodes

Segment  7  Contains  5 Trust Nodes
            Trust Node Is:217
            Trust Node Is:220
            Trust Node Is:202
            Trust Node Is:194
            Trust Node Is:179

Segment  8  Contains  1 Trust Nodes
            Trust Node Is:131

Segment  9  Contains  6 Trust Nodes
            Trust Node Is:7
            Trust Node Is:3
            Trust Node Is:129
            Trust Node Is:72
            Trust Node Is:11
            Trust Node Is:78

Segment  10  Contains  1 Trust Nodes
             Trust Node Is:267

Segment  11  Contains  4 Trust Nodes
             Trust Node Is:28
             Trust Node Is:84
             Trust Node Is:87
             Trust Node Is:86

Segment  12  Contains  1 Trust Nodes
             Trust Node Is:266

Segment  13  Contains  0 Trust Nodes

Segment  14  Contains  1 Trust Nodes
             Trust Node Is:276

Segment  15  Contains  1 Trust Nodes
             Trust Node Is:268


Number Of Trus Nodes: 38
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 249.933333333333
Processing Time:0.7151853
Average Segment Size:20
Coefficient Variation:0.706854197938201
```

```
      Source      Destination      Weight

      31          266              0
      266         270              0
      1           5                0
      2           6                0
      3           129              0
      4           16               0
      7           12               0
      7           110              0
      12          20               0
      16          36               0
      18          20               0
      27          32               0
      28          36               0
      33          36               0
      36          40               0
      51          53               0
      54          123              0
      64          65               0
      81          88               0
      95          99               0
      96          97               0
      97          100              0
      104         105              0
      107         112              0
      108         112              0
      109         111              0
      109         130              0
      112         116              0
      112         147              0
      112         148              0
      112         150              0
      116         119              0
      116         160              0
      116         165              0
      116         167              0
      119         161              0
      136         138              0
      137         138              0
      141         143              0
      141         144              0
      144         145              0
      146         148              0
      160         117              0
      160         166              0
      163         164              0
      169         210              0
      170         171              0
      171         204              0
      173         198              0
      173         242              0
      174         198              0
```

```
Segment  1 Contains  3 Trust Nodes
           Trust Node Is:67
           Trust Node Is:66
           Trust Node Is:73

Segment  2 Contains  3 Trust Nodes
           Trust Node Is:74
           Trust Node Is:185
           Trust Node Is:89

Segment  3 Contains  5 Trust Nodes
           Trust Node Is:35
           Trust Node Is:39
           Trust Node Is:44
           Trust Node Is:37
           Trust Node Is:41

Segment  4 Contains  3 Trust Nodes
           Trust Node Is:119
           Trust Node Is:120
           Trust Node Is:152

Segment  5 Contains  3 Trust Nodes
           Trust Node Is:198
           Trust Node Is:197
           Trust Node Is:191

Segment  6 Contains  1 Trust Nodes
           Trust Node Is:214

Segment  7 Contains  3 Trust Nodes
           Trust Node Is:220
           Trust Node Is:202
           Trust Node Is:179

Segment  8 Contains  3 Trust Nodes
           Trust Node Is:131
           Trust Node Is:14
           Trust Node Is:112

Segment  9 Contains  6 Trust Nodes
           Trust Node Is:7
           Trust Node Is:3
           Trust Node Is:129
           Trust Node Is:72
           Trust Node Is:11
           Trust Node Is:78

Segment  10 Contains  1 Trust Nodes
            Trust Node Is:267
```

Figure 57: The experimental results on IEEE test system of BUS118 with 15 segments.

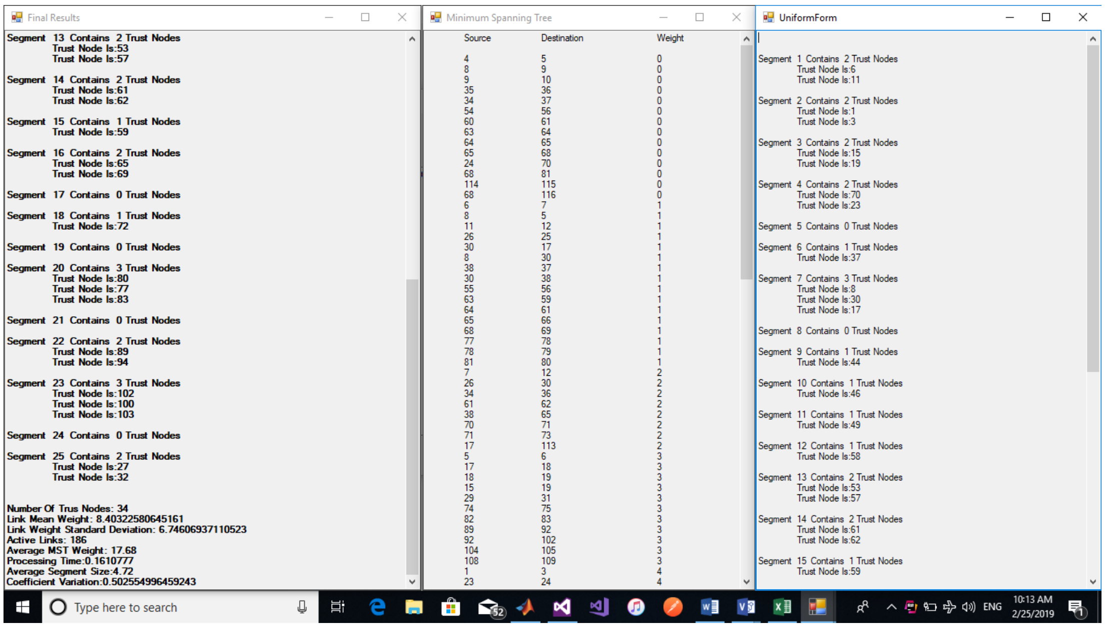


```
            Trust Node Is:46

Segment  9  Contains  2 Trust Nodes
            Trust Node Is:58
            Trust Node Is:57

Segment  10  Contains  2 Trust Nodes
             Trust Node Is:61
             Trust Node Is:62

Segment  11  Contains  1 Trust Nodes
             Trust Node Is:59

Segment  12  Contains  2 Trust Nodes
             Trust Node Is:65
             Trust Node Is:69

Segment  13  Contains  1 Trust Nodes
             Trust Node Is:52

Segment  14  Contains  1 Trust Nodes
             Trust Node Is:72

Segment  15  Contains  0 Trust Nodes

Segment  16  Contains  0 Trust Nodes

Segment  17  Contains  3 Trust Nodes
             Trust Node Is:80
             Trust Node Is:77
             Trust Node Is:83

Segment  18  Contains  4 Trust Nodes
             Trust Node Is:92
             Trust Node Is:94
             Trust Node Is:100
             Trust Node Is:103

Segment  19  Contains  0 Trust Nodes

Segment  20  Contains  2 Trust Nodes
             Trust Node Is:27
             Trust Node Is:32


Number Of Trus Nodes: 32
Link Mean Weight: 8.40322580645161
Link Weight Standard Deviation: 6.74606937110523
Active Links: 186
Average MST Weight: 24.3
Processing Time:0.1292845
Average Segment Size:5.9
Coefficient Variation:0.418432556261778
```

```
      Source      Destination      Weight

      4           5                0
      8           9                0
      9           10               0
      35          36               0
      34          37               0
      54          56               0
      60          61               0
      63          64               0
      64          65               0
      65          68               0
      24          70               0
      68          81               0
      114         115              0
      68          116              0
      6           7                1
      8           5                1
      11          12               1
      26          25               1
      30          17               1
      8           30               1
      38          37               1
      30          38               1
      55          56               1
      63          59               1
      64          61               1
      65          66               1
      68          69               1
      77          78               1
      78          79               1
      81          80               1
      7           12               2
      26          30               2
      34          36               2
      61          62               2
      38          65               2
      70          71               2
      71          73               2
      17          113              2
      5           6                3
      17          18               3
      18          19               3
      15          19               3
      29          31               3
      74          75               3
      82          83               3
      89          92               3
      92          102              3
      104         105              3
      108         109              3
      1           3                4
      23          24               4
```

```
Segment  1 Contains  2 Trust Nodes
           Trust Node Is:6
           Trust Node Is:11
|
Segment  2 Contains  2 Trust Nodes
           Trust Node Is:1
           Trust Node Is:3

Segment  3 Contains  3 Trust Nodes
           Trust Node Is:17
           Trust Node Is:22
           Trust Node Is:15

Segment  4 Contains  2 Trust Nodes
           Trust Node Is:30
           Trust Node Is:70

Segment  5 Contains  2 Trust Nodes
           Trust Node Is:34
           Trust Node Is:43

Segment  6 Contains  0 Trust Nodes

Segment  7 Contains  1 Trust Nodes
           Trust Node Is:37

Segment  8 Contains  2 Trust Nodes
           Trust Node Is:49
           Trust Node Is:46

Segment  9 Contains  2 Trust Nodes
           Trust Node Is:58
           Trust Node Is:57

Segment  10 Contains  2 Trust Nodes
            Trust Node Is:61
            Trust Node Is:62

Segment  11 Contains  1 Trust Nodes
            Trust Node Is:59

Segment  12 Contains  2 Trust Nodes
            Trust Node Is:65
            Trust Node Is:69

Segment  13 Contains  1 Trust Nodes
            Trust Node Is:52

Segment  14 Contains  1 Trust Nodes
            Trust Node Is:72
```

Figure 58: The experimental results on IEEE test system of BUS118 with 20 segments.

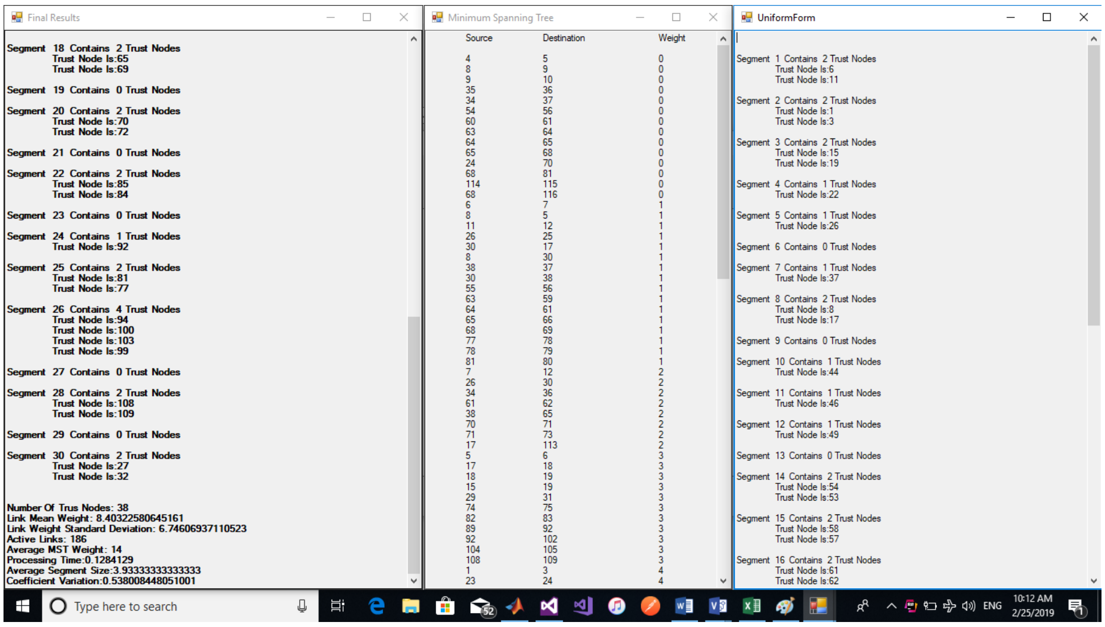


```
Segment  13  Contains  2 Trust Nodes
         Trust Node Is:53
         Trust Node Is:57

Segment  14  Contains  2 Trust Nodes
         Trust Node Is:61
         Trust Node Is:62

Segment  15  Contains  1 Trust Nodes
         Trust Node Is:59

Segment  16  Contains  2 Trust Nodes
         Trust Node Is:65
         Trust Node Is:69

Segment  17  Contains  0 Trust Nodes

Segment  18  Contains  1 Trust Nodes
         Trust Node Is:72

Segment  19  Contains  0 Trust Nodes

Segment  20  Contains  3 Trust Nodes
         Trust Node Is:80
         Trust Node Is:77
         Trust Node Is:83

Segment  21  Contains  0 Trust Nodes

Segment  22  Contains  2 Trust Nodes
         Trust Node Is:89
         Trust Node Is:94

Segment  23  Contains  3 Trust Nodes
         Trust Node Is:102
         Trust Node Is:100
         Trust Node Is:103

Segment  24  Contains  0 Trust Nodes

Segment  25  Contains  2 Trust Nodes
         Trust Node Is:27
         Trust Node Is:32


Number Of Trus Nodes: 34
Link Mean Weight: 8.40322580645161
Link Weight Standard Deviation: 6.74606937110523
Active Links: 186
Average MST Weight: 17.68
Processing Time:0.1610777
Average Segment Size:4.72
Coefficient Variation:0.502554996459243
```

```
Source      Destination      Weight

4           5                0
8           9                0
9           10               0
35          36               0
34          37               0
54          56               0
60          61               0
63          64               0
64          65               0
65          68               0
24          70               0
68          81               0
114         115              0
68          116              0
6           7                1
8           5                1
11          12               1
26          25               1
30          17               1
8           30               1
38          37               1
30          38               1
55          56               1
63          59               1
64          61               1
65          66               1
68          69               1
77          78               1
78          79               1
81          80               1
7           12               2
26          30               2
34          36               2
61          62               2
38          65               2
70          71               2
71          73               2
17          113              2
5           6                3
17          18               3
18          19               3
15          19               3
29          31               3
74          75               3
82          83               3
89          92               3
92          102              3
104         105              3
108         109              3
1           3                4
23          24               4
```

```
|

Segment  1  Contains  2 Trust Nodes
         Trust Node Is:6
         Trust Node Is:11

Segment  2  Contains  2 Trust Nodes
         Trust Node Is:1
         Trust Node Is:3

Segment  3  Contains  2 Trust Nodes
         Trust Node Is:15
         Trust Node Is:19

Segment  4  Contains  2 Trust Nodes
         Trust Node Is:70
         Trust Node Is:23

Segment  5  Contains  0 Trust Nodes

Segment  6  Contains  1 Trust Nodes
         Trust Node Is:37

Segment  7  Contains  3 Trust Nodes
         Trust Node Is:8
         Trust Node Is:30
         Trust Node Is:17

Segment  8  Contains  0 Trust Nodes

Segment  9  Contains  1 Trust Nodes
         Trust Node Is:44

Segment  10  Contains  1 Trust Nodes
         Trust Node Is:46

Segment  11  Contains  1 Trust Nodes
         Trust Node Is:49

Segment  12  Contains  1 Trust Nodes
         Trust Node Is:58

Segment  13  Contains  2 Trust Nodes
         Trust Node Is:53
         Trust Node Is:57

Segment  14  Contains  2 Trust Nodes
         Trust Node Is:61
         Trust Node Is:62

Segment  15  Contains  1 Trust Nodes
         Trust Node Is:59
```

Figure 59: The experimental results on IEEE test system of BUS118 with 25 segments.



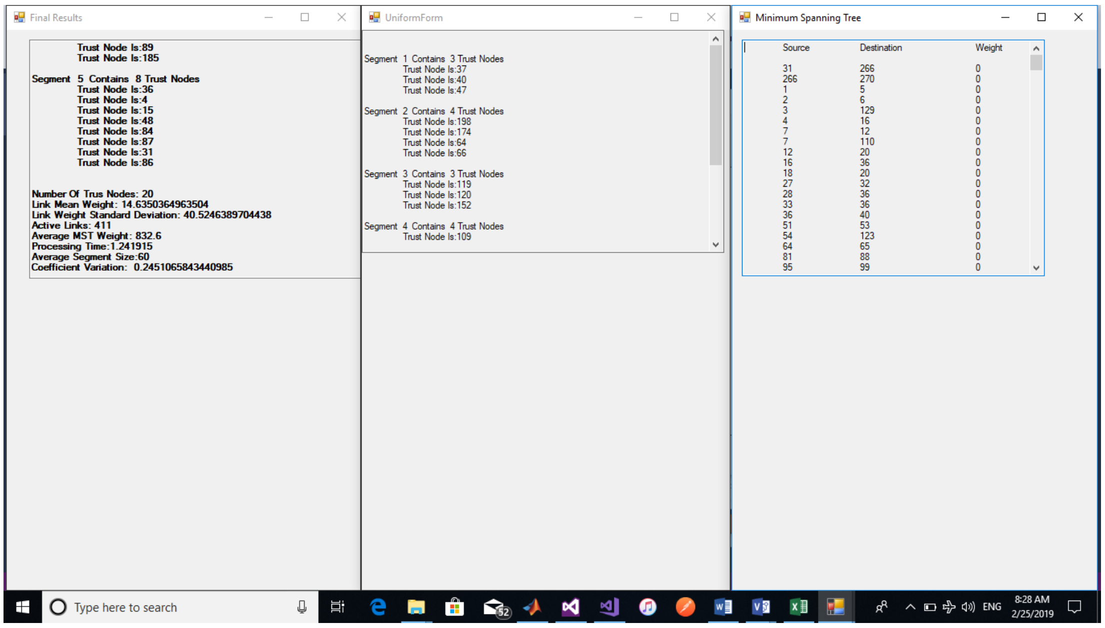
```
Segment  18  Contains  2 Trust Nodes
         Trust Node Is:65
         Trust Node Is:69

Segment  19  Contains  0 Trust Nodes

Segment  20  Contains  2 Trust Nodes
         Trust Node Is:70
         Trust Node Is:72

Segment  21  Contains  0 Trust Nodes

Segment  22  Contains  2 Trust Nodes
         Trust Node Is:85
         Trust Node Is:84

Segment  23  Contains  0 Trust Nodes

Segment  24  Contains  1 Trust Nodes
         Trust Node Is:92

Segment  25  Contains  2 Trust Nodes
         Trust Node Is:81
         Trust Node Is:77

Segment  26  Contains  4 Trust Nodes
         Trust Node Is:94
         Trust Node Is:100
         Trust Node Is:103
         Trust Node Is:99

Segment  27  Contains  0 Trust Nodes

Segment  28  Contains  2 Trust Nodes
         Trust Node Is:108
         Trust Node Is:109

Segment  29  Contains  0 Trust Nodes

Segment  30  Contains  2 Trust Nodes
         Trust Node Is:27
         Trust Node Is:32


Number Of Trus Nodes: 38
Link Mean Weight: 8.40322580645161
Link Weight Standard Deviation: 6.74606937110523
Active Links: 186
Average MST Weight: 14
Processing Time:0.1284129
Average Segment Size:3.93333333333333
Coefficient Variation:0.538008448051001
```

```
Source      Destination      Weight

4           5                0
8           9                0
9           10               0
35          36               0
34          37               0
54          56               0
60          61               0
63          64               0
64          65               0
65          68               0
24          70               0
68          81               0
114         115              0
68          116              0
6           7                1
8           5                1
11          12               1
26          25               1
30          17               1
8           30               1
38          37               1
30          38               1
55          56               1
63          59               1
64          61               1
65          66               1
68          69               1
77          78               1
78          79               1
81          80               1
7           12               2
26          30               2
34          36               2
61          62               2
38          65               2
70          71               2
71          73               2
17          113              2
5           6                3
17          18               3
18          19               3
15          19               3
29          31               3
74          75               3
82          83               3
89          92               3
92          102              3
104         105              3
108         109              3
1           3                4
23          24               4
```

```
|

Segment  1  Contains  2 Trust Nodes
         Trust Node Is:6
         Trust Node Is:11

Segment  2  Contains  2 Trust Nodes
         Trust Node Is:1
         Trust Node Is:3

Segment  3  Contains  2 Trust Nodes
         Trust Node Is:15
         Trust Node Is:19

Segment  4  Contains  1 Trust Nodes
         Trust Node Is:22

Segment  5  Contains  1 Trust Nodes
         Trust Node Is:26

Segment  6  Contains  0 Trust Nodes

Segment  7  Contains  1 Trust Nodes
         Trust Node Is:37

Segment  8  Contains  2 Trust Nodes
         Trust Node Is:8
         Trust Node Is:17

Segment  9  Contains  0 Trust Nodes

Segment  10  Contains  1 Trust Nodes
         Trust Node Is:44

Segment  11  Contains  1 Trust Nodes
         Trust Node Is:46

Segment  12  Contains  1 Trust Nodes
         Trust Node Is:49

Segment  13  Contains  0 Trust Nodes

Segment  14  Contains  2 Trust Nodes
         Trust Node Is:54
         Trust Node Is:53

Segment  15  Contains  2 Trust Nodes
         Trust Node Is:58
         Trust Node Is:57

Segment  16  Contains  2 Trust Nodes
         Trust Node Is:61
         Trust Node Is:62
```

Figure 60: The experimental results on IEEE test system of BUS118 with 30 segments.

Appendix C5: The Experimental Result on IEEE Test System of BUS300:

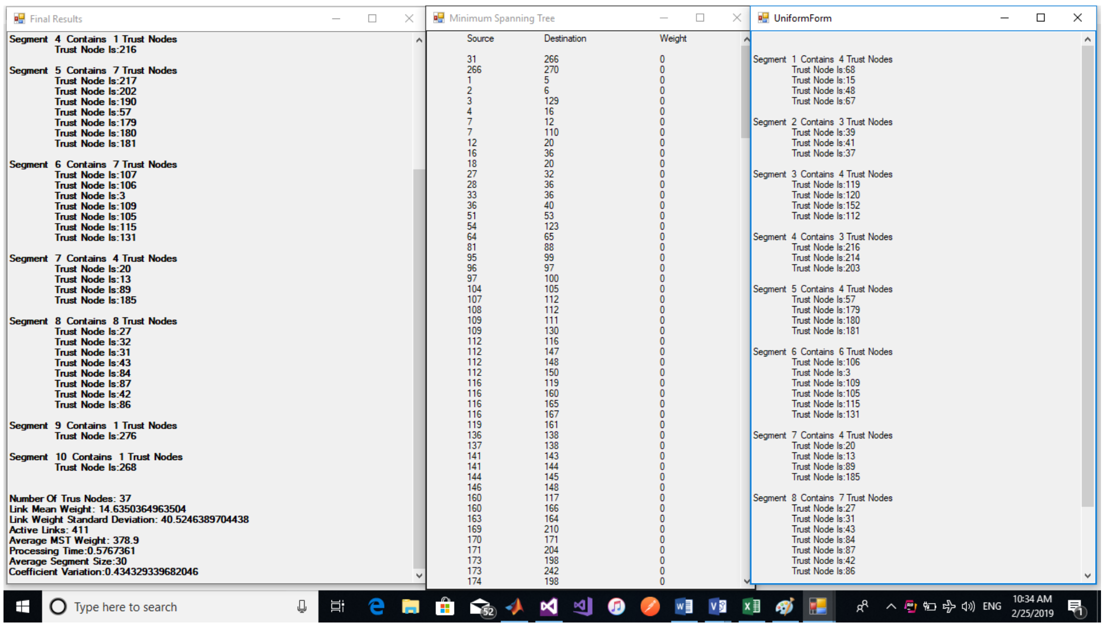


```
                Trust Node Is:89
                Trust Node Is:185

Segment  5  Contains  8 Trust Nodes
                Trust Node Is:36
                Trust Node Is:4
                Trust Node Is:15
                Trust Node Is:48
                Trust Node Is:84
                Trust Node Is:87
                Trust Node Is:31
                Trust Node Is:86


Number Of Trus Nodes: 20
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 832.6
Processing Time:1.241915
Average Segment Size:60
Coefficient Variation:  0.2451065843440985
```

```
Segment  1  Contains  3 Trust Nodes
               Trust Node Is:37
               Trust Node Is:40
               Trust Node Is:47

Segment  2  Contains  4 Trust Nodes
               Trust Node Is:198
               Trust Node Is:174
               Trust Node Is:64
               Trust Node Is:66

Segment  3  Contains  3 Trust Nodes
               Trust Node Is:119
               Trust Node Is:120
               Trust Node Is:152

Segment  4  Contains  4 Trust Nodes
               Trust Node Is:109
```

```
        Source        Destination        Weight

        31            266                0
        266           270                0
        1             5                  0
        2             6                  0
        3             129                0
        4             16                 0
        7             12                 0
        7             110                0
        12            20                 0
        16            36                 0
        18            20                 0
        27            32                 0
        28            36                 0
        33            36                 0
        36            40                 0
        51            53                 0
        54            123                0
        64            65                 0
        81            88                 0
        95            99                 0
```

Figure 61: The experimental results on IEEE test system of BUS300 with 5 segments.

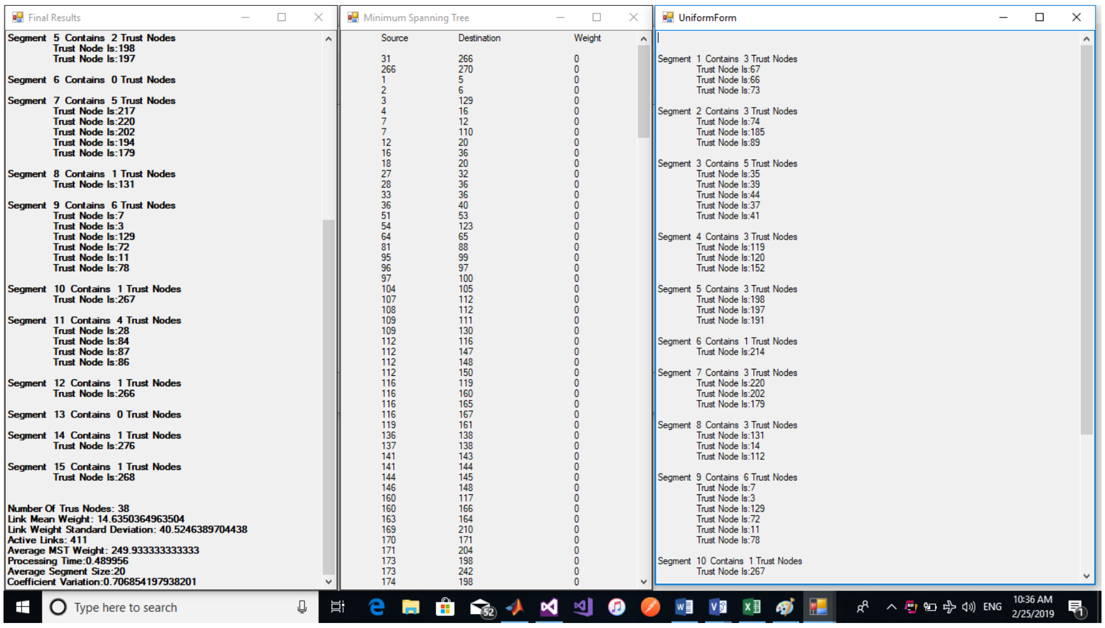


```
Segment  4  Contains  1 Trust Nodes
            Trust Node Is:216

Segment  5  Contains  7 Trust Nodes
            Trust Node Is:217
            Trust Node Is:202
            Trust Node Is:190
            Trust Node Is:57
            Trust Node Is:179
            Trust Node Is:180
            Trust Node Is:181

Segment  6  Contains  7 Trust Nodes
            Trust Node Is:107
            Trust Node Is:106
            Trust Node Is:3
            Trust Node Is:109
            Trust Node Is:105
            Trust Node Is:115
            Trust Node Is:131

Segment  7  Contains  4 Trust Nodes
            Trust Node Is:20
            Trust Node Is:13
            Trust Node Is:89
            Trust Node Is:185

Segment  8  Contains  8 Trust Nodes
            Trust Node Is:27
            Trust Node Is:32
            Trust Node Is:31
            Trust Node Is:43
            Trust Node Is:84
            Trust Node Is:87
            Trust Node Is:42
            Trust Node Is:86

Segment  9  Contains  1 Trust Nodes
            Trust Node Is:276

Segment  10  Contains  1 Trust Nodes
            Trust Node Is:268


Number Of Trus Nodes: 37
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 378.9
Processing Time:0.5767361
Average Segment Size:30
Coefficient Variation:0.434329339682046
```

```
        Source        Destination        Weight

        31            266                0
        266           270                0
        1             5                  0
        2             6                  0
        3             129                0
        4             16                 0
        7             12                 0
        7             110                0
        12            20                 0
        16            36                 0
        18            20                 0
        27            32                 0
        28            36                 0
        33            36                 0
        36            40                 0
        51            53                 0
        54            123                0
        64            65                 0
        81            88                 0
        95            99                 0
        96            97                 0
        97            100                0
        104           105                0
        107           112                0
        108           112                0
        109           111                0
        109           130                0
        112           116                0
        112           147                0
        112           148                0
        112           150                0
        116           119                0
        116           160                0
        116           165                0
        116           167                0
        119           161                0
        136           138                0
        137           138                0
        141           143                0
        141           144                0
        144           145                0
        146           148                0
        160           117                0
        160           166                0
        163           164                0
        169           210                0
        170           171                0
        171           204                0
        173           198                0
        173           242                0
        174           198                0
```

```
Segment  1  Contains  4 Trust Nodes
             Trust Node Is:68
             Trust Node Is:15
             Trust Node Is:48
             Trust Node Is:67

Segment  2  Contains  3 Trust Nodes
             Trust Node Is:39
             Trust Node Is:41
             Trust Node Is:37

Segment  3  Contains  4 Trust Nodes
             Trust Node Is:119
             Trust Node Is:120
             Trust Node Is:152
             Trust Node Is:112

Segment  4  Contains  3 Trust Nodes
             Trust Node Is:216
             Trust Node Is:214
             Trust Node Is:203

Segment  5  Contains  4 Trust Nodes
             Trust Node Is:57
             Trust Node Is:179
             Trust Node Is:180
             Trust Node Is:181

Segment  6  Contains  6 Trust Nodes
             Trust Node Is:106
             Trust Node Is:3
             Trust Node Is:109
             Trust Node Is:105
             Trust Node Is:115
             Trust Node Is:131

Segment  7  Contains  4 Trust Nodes
             Trust Node Is:20
             Trust Node Is:13
             Trust Node Is:89
             Trust Node Is:185

Segment  8  Contains  7 Trust Nodes
             Trust Node Is:27
             Trust Node Is:31
             Trust Node Is:43
             Trust Node Is:84
             Trust Node Is:87
             Trust Node Is:42
             Trust Node Is:86
```

Figure 62: The experimental results on IEEE test system of BUS300 with 10 segments.

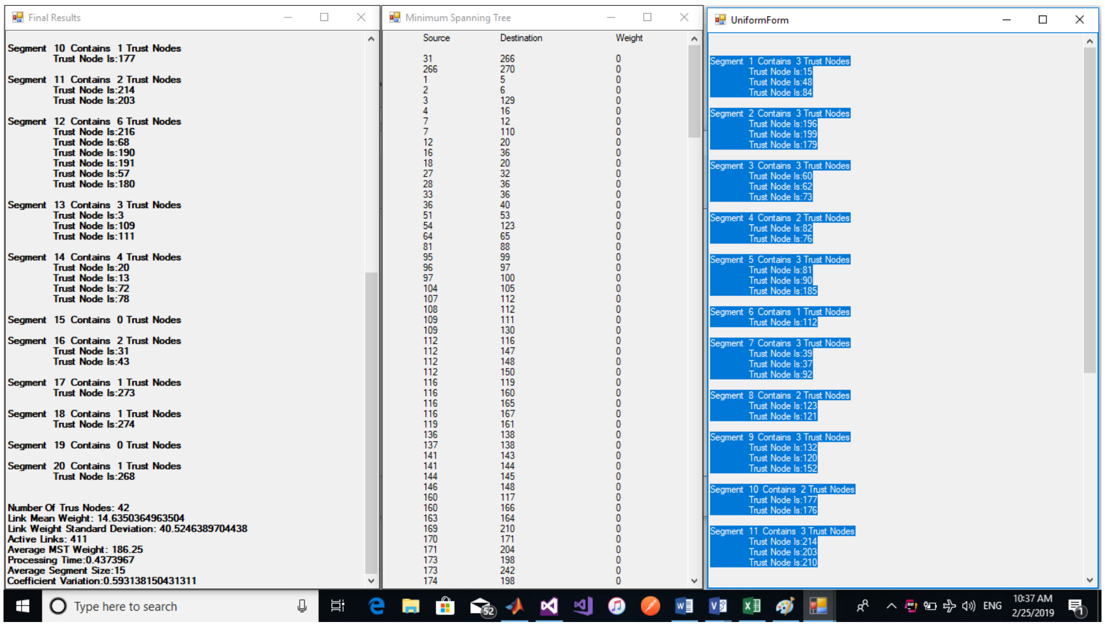


```
Segment  5  Contains  2 Trust Nodes
            Trust Node Is:198
            Trust Node Is:197

Segment  6  Contains  0 Trust Nodes

Segment  7  Contains  5 Trust Nodes
            Trust Node Is:217
            Trust Node Is:220
            Trust Node Is:202
            Trust Node Is:194
            Trust Node Is:179

Segment  8  Contains  1 Trust Nodes
            Trust Node Is:131

Segment  9  Contains  6 Trust Nodes
            Trust Node Is:7
            Trust Node Is:3
            Trust Node Is:129
            Trust Node Is:72
            Trust Node Is:11
            Trust Node Is:78

Segment  10  Contains  1 Trust Nodes
             Trust Node Is:267

Segment  11  Contains  4 Trust Nodes
             Trust Node Is:28
             Trust Node Is:84
             Trust Node Is:87
             Trust Node Is:86

Segment  12  Contains  1 Trust Nodes
             Trust Node Is:266

Segment  13  Contains  0 Trust Nodes

Segment  14  Contains  1 Trust Nodes
             Trust Node Is:276

Segment  15  Contains  1 Trust Nodes
             Trust Node Is:268


Number Of Trus Nodes: 38
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 249.933333333333
Processing Time:0.489956
Average Segment Size:20
Coefficient Variation:0.706854197938201
```

```
     Source      Destination     Weight

     31          266             0
     266         270             0
     1           5               0
     2           6               0
     3           129             0
     4           16              0
     7           12              0
     7           110             0
     12          20              0
     16          36              0
     18          20              0
     27          32              0
     28          36              0
     33          36              0
     36          40              0
     51          53              0
     54          123             0
     64          65              0
     81          88              0
     95          99              0
     96          97              0
     97          100             0
     104         105             0
     107         112             0
     108         112             0
     109         111             0
     109         130             0
     112         116             0
     112         147             0
     112         148             0
     112         150             0
     116         119             0
     116         160             0
     116         165             0
     116         167             0
     119         161             0
     136         138             0
     137         138             0
     141         143             0
     141         144             0
     144         145             0
     146         148             0
     160         117             0
     160         166             0
     163         164             0
     169         210             0
     170         171             0
     171         204             0
     173         198             0
     173         242             0
     174         198             0
```

```
|
Segment  1  Contains  3 Trust Nodes
            Trust Node Is:67
            Trust Node Is:66
            Trust Node Is:73

Segment  2  Contains  3 Trust Nodes
            Trust Node Is:74
            Trust Node Is:185
            Trust Node Is:89

Segment  3  Contains  5 Trust Nodes
            Trust Node Is:35
            Trust Node Is:39
            Trust Node Is:44
            Trust Node Is:37
            Trust Node Is:41

Segment  4  Contains  3 Trust Nodes
            Trust Node Is:119
            Trust Node Is:120
            Trust Node Is:152

Segment  5  Contains  3 Trust Nodes
            Trust Node Is:198
            Trust Node Is:197
            Trust Node Is:191

Segment  6  Contains  1 Trust Nodes
            Trust Node Is:214

Segment  7  Contains  3 Trust Nodes
            Trust Node Is:220
            Trust Node Is:202
            Trust Node Is:179

Segment  8  Contains  3 Trust Nodes
            Trust Node Is:131
            Trust Node Is:14
            Trust Node Is:112

Segment  9  Contains  6 Trust Nodes
            Trust Node Is:7
            Trust Node Is:3
            Trust Node Is:129
            Trust Node Is:72
            Trust Node Is:11
            Trust Node Is:78

Segment  10  Contains  1 Trust Nodes
             Trust Node Is:267
```

Figure 63: The experimental results on IEEE test system of BUS300 with 15 segments.

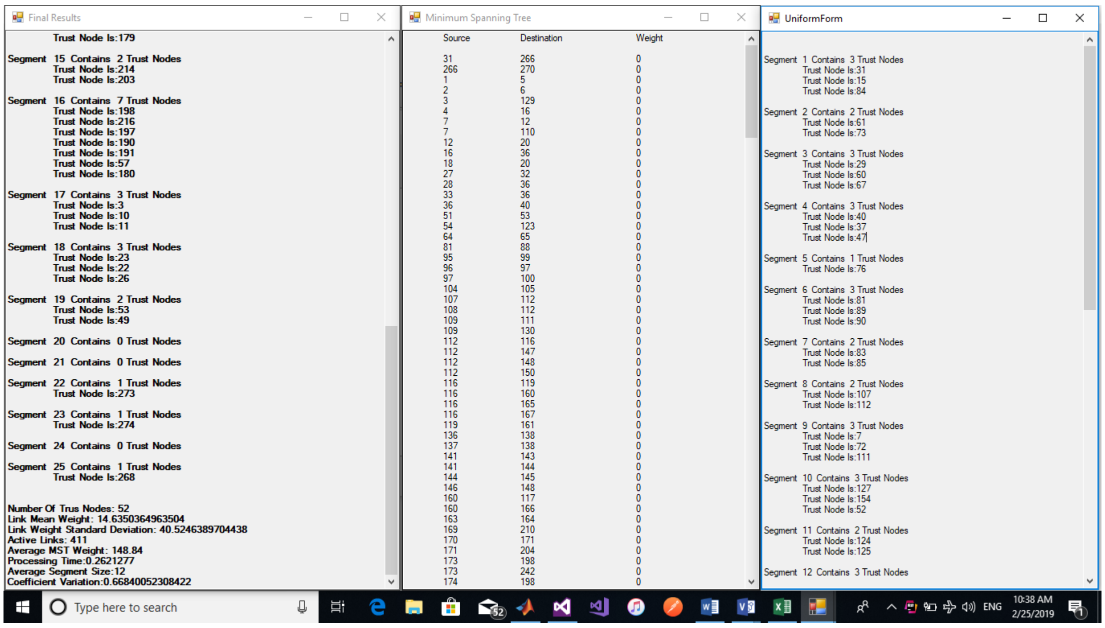


```
Segment  10  Contains  1 Trust Nodes
             Trust Node Is:177

Segment  11  Contains  2 Trust Nodes
             Trust Node Is:214
             Trust Node Is:203

Segment  12  Contains  6 Trust Nodes
             Trust Node Is:216
             Trust Node Is:68
             Trust Node Is:190
             Trust Node Is:191
             Trust Node Is:57
             Trust Node Is:180

Segment  13  Contains  3 Trust Nodes
             Trust Node Is:3
             Trust Node Is:109
             Trust Node Is:111

Segment  14  Contains  4 Trust Nodes
             Trust Node Is:20
             Trust Node Is:13
             Trust Node Is:72
             Trust Node Is:78

Segment  15  Contains  0 Trust Nodes

Segment  16  Contains  2 Trust Nodes
             Trust Node Is:31
             Trust Node Is:43

Segment  17  Contains  1 Trust Nodes
             Trust Node Is:273

Segment  18  Contains  1 Trust Nodes
             Trust Node Is:274

Segment  19  Contains  0 Trust Nodes

Segment  20  Contains  1 Trust Nodes
             Trust Node Is:268


Number Of Trus Nodes: 42
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 186.25
Processing Time:0.4373967
Average Segment Size:15
Coefficient Variation:0.593138150431311
```

```
     Source      Destination     Weight

     31          266             0
     266         270             0
     1           5               0
     2           6               0
     3           129             0
     4           16              0
     7           12              0
     7           110             0
     12          20              0
     16          36              0
     18          20              0
     27          32              0
     28          36              0
     33          36              0
     36          40              0
     51          53              0
     54          123             0
     64          65              0
     81          88              0
     95          99              0
     96          97              0
     97          100             0
     104         105             0
     107         112             0
     108         112             0
     109         111             0
     109         130             0
     112         116             0
     112         147             0
     112         148             0
     112         150             0
     116         119             0
     116         160             0
     116         165             0
     116         167             0
     119         161             0
     136         138             0
     137         138             0
     141         143             0
     141         144             0
     144         145             0
     146         148             0
     160         117             0
     160         166             0
     163         164             0
     169         210             0
     170         171             0
     171         204             0
     173         198             0
     173         242             0
     174         198             0
```

```
Segment  1  Contains  3 Trust Nodes
            Trust Node Is:15
            Trust Node Is:48
            Trust Node Is:84

Segment  2  Contains  3 Trust Nodes
            Trust Node Is:196
            Trust Node Is:199
            Trust Node Is:179

Segment  3  Contains  3 Trust Nodes
            Trust Node Is:60
            Trust Node Is:62
            Trust Node Is:73

Segment  4  Contains  2 Trust Nodes
            Trust Node Is:82
            Trust Node Is:76

Segment  5  Contains  3 Trust Nodes
            Trust Node Is:81
            Trust Node Is:90
            Trust Node Is:185

Segment  6  Contains  1 Trust Nodes
            Trust Node Is:112

Segment  7  Contains  3 Trust Nodes
            Trust Node Is:39
            Trust Node Is:37
            Trust Node Is:92

Segment  8  Contains  2 Trust Nodes
            Trust Node Is:123
            Trust Node Is:121

Segment  9  Contains  3 Trust Nodes
            Trust Node Is:132
            Trust Node Is:120
            Trust Node Is:152

Segment  10  Contains  2 Trust Nodes
             Trust Node Is:177
             Trust Node Is:176

Segment  11  Contains  3 Trust Nodes
             Trust Node Is:214
             Trust Node Is:203
             Trust Node Is:210
```

Figure 64: The experimental results on IEEE test system of BUS300 with 20 segments.

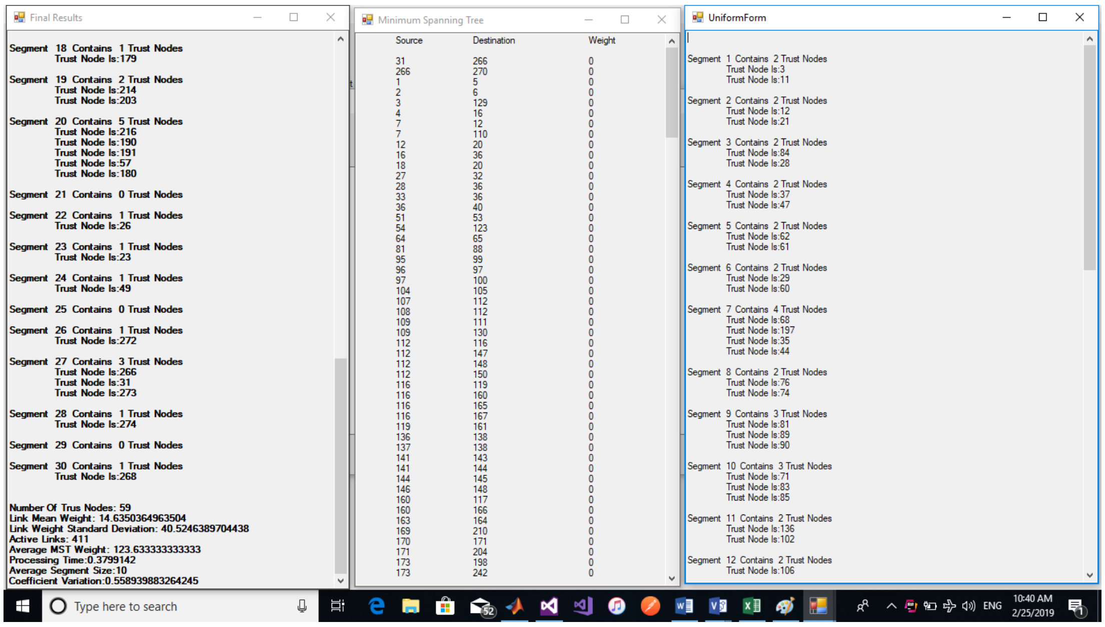


```
Final Results

              Trust Node Is:179

Segment  15  Contains  2 Trust Nodes
              Trust Node Is:214
              Trust Node Is:203

Segment  16  Contains  7 Trust Nodes
              Trust Node Is:198
              Trust Node Is:216
              Trust Node Is:197
              Trust Node Is:190
              Trust Node Is:191
              Trust Node Is:57
              Trust Node Is:180

Segment  17  Contains  3 Trust Nodes
              Trust Node Is:3
              Trust Node Is:10
              Trust Node Is:11

Segment  18  Contains  3 Trust Nodes
              Trust Node Is:23
              Trust Node Is:22
              Trust Node Is:26

Segment  19  Contains  2 Trust Nodes
              Trust Node Is:53
              Trust Node Is:49

Segment  20  Contains  0 Trust Nodes

Segment  21  Contains  0 Trust Nodes

Segment  22  Contains  1 Trust Nodes
              Trust Node Is:273

Segment  23  Contains  1 Trust Nodes
              Trust Node Is:274

Segment  24  Contains  0 Trust Nodes

Segment  25  Contains  1 Trust Nodes
              Trust Node Is:268


Number Of Trus Nodes: 52
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 148.84
Processing Time:0.2621277
Average Segment Size:12
Coefficient Variation:0.66840052308422
```

```
Minimum Spanning Tree

      Source      Destination      Weight

      31          266              0
      266         270              0
      1           5                0
      2           6                0
      3           129              0
      4           16               0
      7           12               0
      7           110              0
      12          20               0
      16          36               0
      18          20               0
      27          32               0
      28          36               0
      33          36               0
      36          40               0
      51          53               0
      54          123              0
      64          65               0
      81          88               0
      95          99               0
      96          97               0
      97          100              0
      104         105              0
      107         112              0
      108         112              0
      109         111              0
      109         130              0
      112         116              0
      112         147              0
      112         148              0
      112         150              0
      116         119              0
      116         160              0
      116         165              0
      116         167              0
      119         161              0
      136         138              0
      137         138              0
      141         143              0
      141         144              0
      144         145              0
      146         148              0
      160         117              0
      160         166              0
      163         164              0
      169         210              0
      170         171              0
      171         204              0
      173         198              0
      173         242              0
      174         198              0
```

```
UniformForm


Segment  1  Contains  3 Trust Nodes
           Trust Node Is:31
           Trust Node Is:15
           Trust Node Is:84

Segment  2  Contains  2 Trust Nodes
           Trust Node Is:61
           Trust Node Is:73

Segment  3  Contains  3 Trust Nodes
           Trust Node Is:29
           Trust Node Is:60
           Trust Node Is:67

Segment  4  Contains  3 Trust Nodes
           Trust Node Is:40
           Trust Node Is:37
           Trust Node Is:47

Segment  5  Contains  1 Trust Nodes
           Trust Node Is:76

Segment  6  Contains  3 Trust Nodes
           Trust Node Is:81
           Trust Node Is:89
           Trust Node Is:90

Segment  7  Contains  2 Trust Nodes
           Trust Node Is:83
           Trust Node Is:85

Segment  8  Contains  2 Trust Nodes
           Trust Node Is:107
           Trust Node Is:112

Segment  9  Contains  3 Trust Nodes
           Trust Node Is:7
           Trust Node Is:72
           Trust Node Is:111

Segment  10  Contains  3 Trust Nodes
           Trust Node Is:127
           Trust Node Is:154
           Trust Node Is:52

Segment  11  Contains  2 Trust Nodes
           Trust Node Is:124
           Trust Node Is:125

Segment  12  Contains  3 Trust Nodes
```

Figure 65: The experimental results on IEEE test system of BUS300 with 25 segments.



```
Final Results

Segment  18  Contains  1 Trust Nodes
              Trust Node Is:179

Segment  19  Contains  2 Trust Nodes
              Trust Node Is:214
              Trust Node Is:203

Segment  20  Contains  5 Trust Nodes
              Trust Node Is:216
              Trust Node Is:190
              Trust Node Is:191
              Trust Node Is:57
              Trust Node Is:180

Segment  21  Contains  0 Trust Nodes

Segment  22  Contains  1 Trust Nodes
              Trust Node Is:26

Segment  23  Contains  1 Trust Nodes
              Trust Node Is:23

Segment  24  Contains  1 Trust Nodes
              Trust Node Is:49

Segment  25  Contains  0 Trust Nodes

Segment  26  Contains  1 Trust Nodes
              Trust Node Is:272

Segment  27  Contains  3 Trust Nodes
              Trust Node Is:266
              Trust Node Is:31
              Trust Node Is:273

Segment  28  Contains  1 Trust Nodes
              Trust Node Is:274

Segment  29  Contains  0 Trust Nodes

Segment  30  Contains  1 Trust Nodes
              Trust Node Is:268


Number Of Trus Nodes: 59
Link Mean Weight: 14.6350364963504
Link Weight Standard Deviation: 40.5246389704438
Active Links: 411
Average MST Weight: 123.633333333333
Processing Time:0.3799142
Average Segment Size:10
Coefficient Variation:0.558939883264245
```

```
Minimum Spanning Tree

      Source      Destination      Weight

      31          266              0
      266         270              0
      1           5                0
      2           6                0
      3           129              0
      4           16               0
      7           12               0
      7           110              0
      12          20               0
      16          36               0
      18          20               0
      27          32               0
      28          36               0
      33          36               0
      36          40               0
      51          53               0
      54          123              0
      64          65               0
      81          88               0
      95          99               0
      96          97               0
      97          100              0
      104         105              0
      107         112              0
      108         112              0
      109         111              0
      109         130              0
      112         116              0
      112         147              0
      112         148              0
      112         150              0
      116         119              0
      116         160              0
      116         165              0
      116         167              0
      119         161              0
      136         138              0
      137         138              0
      141         143              0
      141         144              0
      144         145              0
      146         148              0
      160         117              0
      160         166              0
      163         164              0
      169         210              0
      170         171              0
      171         204              0
      173         198              0
      173         242              0
```

```
UniformForm

Segment  1  Contains  2 Trust Nodes
           Trust Node Is:3
           Trust Node Is:11

Segment  2  Contains  2 Trust Nodes
           Trust Node Is:12
           Trust Node Is:21

Segment  3  Contains  2 Trust Nodes
           Trust Node Is:84
           Trust Node Is:28

Segment  4  Contains  2 Trust Nodes
           Trust Node Is:37
           Trust Node Is:47

Segment  5  Contains  2 Trust Nodes
           Trust Node Is:62
           Trust Node Is:61

Segment  6  Contains  2 Trust Nodes
           Trust Node Is:29
           Trust Node Is:60

Segment  7  Contains  4 Trust Nodes
           Trust Node Is:68
           Trust Node Is:197
           Trust Node Is:35
           Trust Node Is:44

Segment  8  Contains  2 Trust Nodes
           Trust Node Is:76
           Trust Node Is:74

Segment  9  Contains  3 Trust Nodes
           Trust Node Is:81
           Trust Node Is:89
           Trust Node Is:90

Segment  10  Contains  3 Trust Nodes
           Trust Node Is:71
           Trust Node Is:83
           Trust Node Is:85

Segment  11  Contains  2 Trust Nodes
           Trust Node Is:136
           Trust Node Is:102

Segment  12  Contains  2 Trust Nodes
           Trust Node Is:106
```

Figure 66: The experimental results on IEEE test system of BUS300 with 30 segments.