5-1-2015

Situational Assessment using graph comparison

Pavan Kumar Pallapunidi
University of Nevada, Las Vegas, pallapun@unlv.nevada.edu

SITUATIONAL ASSESSMENT USING GRAPH COMPARISON

By

Pavan Kumar Pallapunidi

Bachelor of Technology, Information Technology

Jawaharlal Nehru Technological University, India

2010

A thesis submitted in partial fulfillment of the requirements

for the

Master of Science - Computer Science

Department of Computer Science

Howard R. Hughes College of Engineering

The Graduate College

University of Nevada, Las Vegas

May 2015

UNLV | GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Pallapunidi Pavan Kumar

entitled

Situational Assessment Using Graph Comparison

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science

Wolfgang Bein, Ph.D., Committee Chair

Ju-Yeon Jo, Ph.D., Committee Member

Ajoy K. Datta, Ph.D., Committee Member

Venkatesan Muthukumar , Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

May 2015

ABSTRACT

Situational Assessment using Graph Comparison

by

Pallapunidi Pavan Kumar

Dr. Wolfgang Bein, Examination Committee Chair
Professor, Department of Computer Science
University of Nevada, Las Vegas.

In strategic operations, the assessment of any given situation is very important and may trigger the development of a mission plan. The mission plan consists of various actions that should be executed in order to successfully mitigate the situation. For a new mission plan to be designed or implemented, the effect of the previous mission plan should be accessed. These mission plans use various sensors to collect the data which can be very large and aggregate them to obtain detailed information of the situation. In order to implement an effective mission plan the current situation has to be assessed

effectively. We propose to model the situation as a graph in which the nodes denote the participants and edges denotes relationships between participants.

Situational assessment for a given situation consists of identifying the current participants and the relationships between current participants. We model these participants as vertices of a graph and the relationships between the participants as weighted arcs. As events happen the situation changes, so does the graph. Changes in the graph can be dramatical or negligible. We derive the similarity between the two graphs at different moments of time. By doing so we will be able to see the effect of the event that caused the change in the graph structure.

We are comparing the similarities of the graphs using the concept of minimum spanning tree. The minimum spanning tree of a graph is a rough estimate of the details of the nodes and the edges of the graph. We therefore propose a new way of assessing a situation and a new way of analyzing the differences between the same set of participants at various intervals of time.

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Wolfgang Bein for inspiring , motivating and supporting me in successful completion of my thesis. His support was always there during my difficult times and constantly motivated me. I would like to extend my thankfulness to Dr. Doina Bein for her guidance whenever I required.

It has been an honor to have Dr. Ajoy K. Datta, Dr. Ju-Yeon Jo, Dr. Venkatesan Muthukumar in this committee and my sincere thanks to all of them.

I would like to specially thank Dr. Ajoy K. Datta for his support throughout my masters degree at UNLV. He is the first person I go to when in need and he guided me through the right path. I cannot say enough words to appreciate his help.

I would like to take this opportunity to convey my deepest gratitude to my parents Mr. P Ranga Rao, Mrs. Pushpa, my uncle and aunt Mr. Venu Gopal and Mrs. Vijaya Lakshmi, my brother Durga Kiran and all my friends for all their support and love in every walk of my life.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the scenarios like military operations, assessing the condition of the enemy camp is very important. To make a plan for handling the situation effectively, the participants in the situation and the relationships between them (such as communication methods, relative position) are to be discovered and assessed. Getting the accurate picture of the current situation is an important milestone in deciding which action to be taken among all the possible actions and also prioritizing the tasks that have to be handled once such an action is to be executed. Once each task is executed we would want to assess the situation again to prioritize the tasks and also to know the effect of the previous action.

For example let us consider two competing teams were involved in a military operation, generally referred as blue team and the red team. The goal of every team is to design plans and execute them to destroy other team's properties and render its communication channels useless. The best plan that maximizes the destruction of the opposite camp would be opted for the final execution. In this stage the entire operation will be monitored by the personnel to analyze the effect of the executions performed so far. This situation analysis is very important, because the forthcoming plans have to be

improved or altered according to the analysis from the monitoring personnel. This analysis may include comparisons between the planned results and the actual results of the actions. The deviation of actual results from the planned results describes the effect of the action taken. In general this analysis should answer questions like: Are we doing the right things? Are we doing the things right?

Lanchester was the first researcher that developed the theory of MOE[15]. It was dependent on differential equation based modeling of the World War I air craft combat. Basing on the work by Lanchester, Brown, Washburn & Kress and Hester & Tolk added stochastic behaviors to account for random and unpredictable behaviors of the military operations. Washburn & Kress developed Continuous Time Markov Chain(CTMC) model. This is two dimensional. In this the present state or condition is defined by a pair (m,n), in which m denotes the number of assets owned by a blue force and n is the number of assets owned by the red force. In this model for example a transition from (m,n) to (m,n-1) indicate there is a loss of 1 asset in the red force and the no of assets of blue force is same compared to the previous situation. In the same way if the transition is from (m,n) to (m-1,n) indicate there is a loss of 1 asset in blue force and no difference in the number of assets owned by red force. The transition from (m,n) to (m',0) indicates the victory of blue force because the no of assets currently in the red force is zero and there is difference in the number of assets of blue force and the transition from (m,n) to (0,n') indicate the victory of red force because the number of assets currently in blue force is zero and there are few number of assets remaining in red force.

1.1 Motivation

The above mentioned model assumes that all the assets in the situation are identical like, all are soldiers or tanks or anything. But all are identical and hence a total number m or n could give the analysis of situation. But in the real war case scenario it is not so. There can be combinations of many different items. For the example instead of considering m identical items in blue force, we should consider there are M types and some specific number of each type like m_1, m_2,..... In the same way instead of considering n identical items in red force, we should consider there are N types and some specific number of each type like n_1, n_2,..... This real case scenario cannot be handled by the model described above. It has to be extended to a 3 dimensional model by considering a 2xMxN matrix for each state. This would increase the calculations and complexity enormously.

In military missions, time is also an important factor to be considered. If M and N are very large (for example in the order of thousands) the time used for calculations and the space complexity also will be very high. Accordingly the time taken to design an effective mission plan is also very large. Even if an effective mission plan is designed, if it is not executed with in time constraints, the effect is reduced drastically or negligible depending on the condition. We do not want a mission plan that has negligible effect. So we have to increase the quality of the process being followed or design a new, faster way of analyzing the data.

To reduce the complexities, calculations and assessing the situation, we propose this new model based on graph comparisons. This model assumes that the objects in the situation to be modeled as nodes and relationships between them to be modeled as the edges connecting these nodes. The situations will be compared depending on the minimum spanning trees of graphs got from both the situations i.e. same situation at different times.

For example let us consider a military war field scenario.

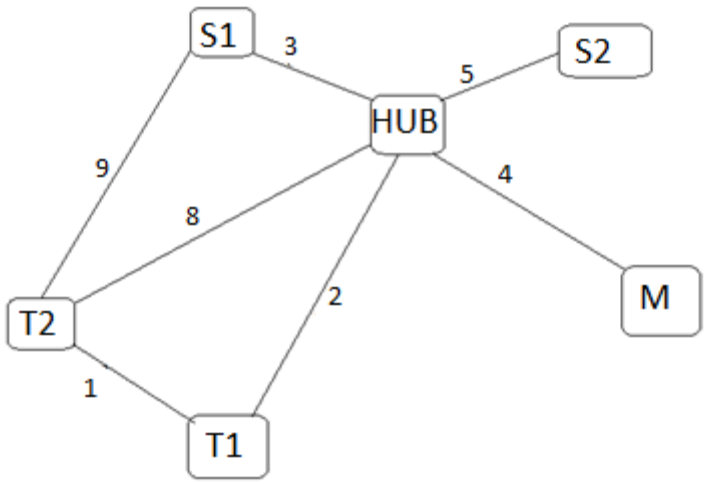

Figure 4.1: An example for a military scenario.

In the figure 1.1 let us consider Hub as the central repository for collecting the data. T1 and T2 are the tanks in the war field connected to each other using communication channel. They are also connected to the hub transferring the data in both the directions. S1 and S2 are the soldiers in the battle field connected to the hub and S1 is

also in connection with T2. There is a monitor for the entire process who is also connected to the hub. We also assume that the number present on each of the edge is the time taken for the communication to transfer between their nodes. This is the situation at enemy's camp. Now we have to design a mission plan which effects the situation.

Let us assume that a mission plan is executed, and we collected the details of the situation again and modeled as the graph. The new graph is as follows.

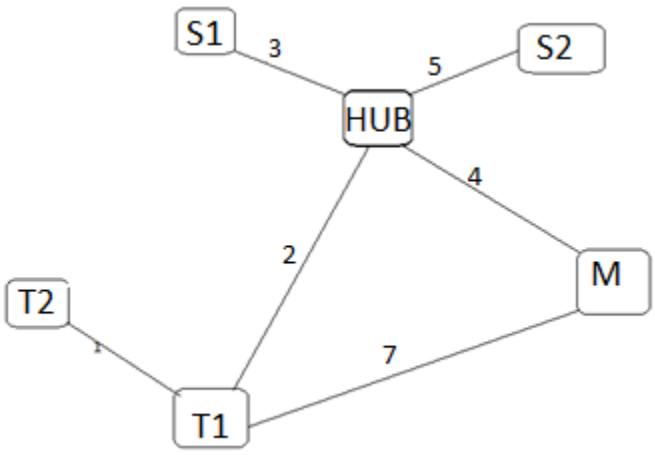

Figure 1.2: Result in scenario after mission plan A.

We observe 3 differences in the new graph compared to the old graph.

- The connection between T2 and Hub is lost.
- The connection between T2 and S1 is lost.
- A new connection between T1 and M.

The graphs for both the situations are different. But the mission had no effect because there was no change in the minimum spanning tree for both the cases. The communication from T2 will be T2-T1-Hub. In both the cases the length of minimum spanning tree is 15 and the edges of minimum spanning tree are T2-T1, T1-Hub, S1-Hub, S2-Hub, M-Hub. So the mission failed in handling the situation.

If the graph that is modeled for the situation after the mission is

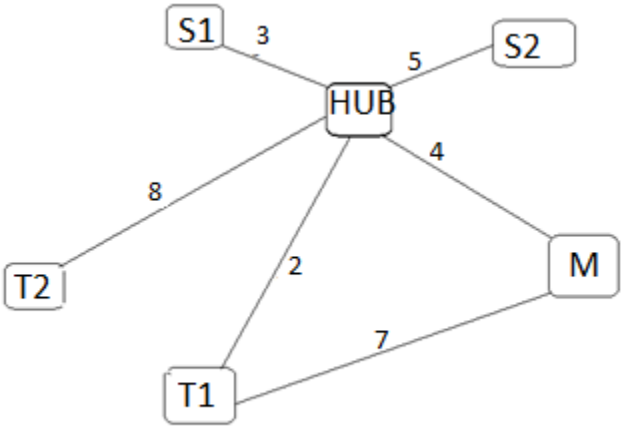

Figure 1.3: Result in scenario after mission plan B.

We observe 3 differences compared to the original graph.

- The connection between T2 and T1 is lost.
- The connection between T2 and S1 is lost.
- A new connection between T1 and M.

In this case the mission is successful in handling the situation. There is a difference in the spanning tree of the graph. Now the total weight of the minimum spanning tree is 22 which was 15 in the previous situation. The new connection between T1 and M, the lost connection between T2 and S1 did not affect the mission.

1.2 Related Work

The comparison of readability of two graph representations i.e. matrix based representations and node link based diagrams is addressed in [1]. The evaluation in this model is based on seven generic tasks. It also recommends the representation depending on the size and density of the graphs involved in the comparison. The node link representation is very familiar as it is often used in representing the graphs. It concludes that, the matrix representation gives better results if the size is more. For example, if a graph has large number of vertices like twenty or more, matrix representation gives better performance over node link diagrams. The matrix based representation has quick layout and superior readability with regard to many tasks. A wider use of this representation will result in it being more familiar. For the real time monitoring, where the graphs evolve dynamically, the matrix representations seem to be very helpful.

A graph type called maximum common subgraph is discussed in [2]. A graph can be called a maximum common subgraph g of two graphs A and B, if there exists no other subgraph of those two graphs A and B that has more number of nodes than g. Graphs are very powerful and versatile. In applications like pattern recognition, the objects can be represented as graphs. Comparison between the structured objects can be obtained from

comparing the graphs of these structured objects. This paper proposes the comparison between two exact algorithms that can be used to find the maximum common subgraph. One is Space State Search algorithm for finding the maximum common subgraph in which all the sub graphs of the given two graphs are taken and the one with high value is chosen as the maximum common sub graph. The other is deriving the maximum common subgraph by first obtaining the association of the given two graphs and then detecting using maximum clique of the second graph. The algorithm required for a purpose is randomly selected from the set of data input values. So in order to compare the two algorithms, a large database of randomly selected sets were used to compare all the possibilities In order to compare the two algorithms, their implementations were generated in C++. The results from the implementation concluded that for the graphs with low density, the first algorithm is more convenient. But for the high edge density, the second algorithm is more favorable.

The algorithm to find the largest approximately common substructure for trees is described in [3]. Finding the common substructure when comparing two trees is similar to the common subgraph discussed in [2], except for the edit distance between the trees being compared comes to play. The common substructure for trees T1 and T2 are the substructures S1 and S2, if S1 is in edit distance d from S2 and there is no other substructure that is greater than S1 and S2 and matches the edit distance constraint. The algorithm discussed here uses the dynamic programming.

The importance of situational assessment reasoning and user awareness of the situation is addressed in [4]. The differences that stressed up between the sensor management and fusion with the revisions in JDL model were addressed in this work.

This paper also highlights role of the user in managing the system and control, assessing the quality of information i.e. metrics to support situation awareness, evaluation of fusion systems to deliver user info needs, planning delivery of knowledge for updating dynamically, designing the interfaces of the SA to support user reasoning.

A method of forming synthetic aperture radar images (SAR) of the moving targets without having any knowledge about the movement of the targets is discussed in [5]. A method called keystone formatting is also discussed in which a unique kernel processing is used involving a one-dimensional interpolation. This preprocessing removes the effects of the linear range migration of the moving targets even without any knowledge on the velocity of the target moving. The quadratic range migration errors will be removed by involving two dimensional focusing of the moving targets.

Tracking of multiple moving targets by using a mobile robot is presented in [6]. Particle filters and statistical data association are used for this purpose. A sample based variant of joint probabilistic data association filters were introduced for tracking the features originating from individual objects. It is also used to solve the correspondence problem between the detected features. Even if the trajectories of the moving targets cross each other, this method will be able to handle the situation.

A new graph similarity calculation procedure to compare the labeled graphs is introduced in [7]. This method initially screens to determine if it is possible to measure the similarity between the two graphs and if it exceeds the given minimum similarity threshold. After the initial screening process, rigorous maximum common edge subgraph (MCES) detection algorithms follows to compute the degree and similarity. A new MCES algorithm is also proposed. The new algorithm is based on maximum clique

formulation of the problem. It also presents new approaches to both lower and upper bounding as well as vertex location.

The graph search based on structural similarity is discussed in [8]. There are many network components in the real world that can be represented as graphs. For example consider a road network. All the intersections of the roads can be considered as vertices of the graph and the roads connecting the vertices as the edges of the graph. The authors proposed an effective graph manipulation techniques to use them in the graph similarity searches. Given a graph G1, if we want to find the graphs that are similar to G1 in the database available with respect to similarity measure. After studying the similarity measure between the two graphs, by using multi dimensional vectors, the graph representation techniques will be discussed. Finally they illustrate representative queries that are handled by the approach proposed. The results also showed that saving of computational effort when compared with the traditional searching algorithms.

All the details related to a situation are collected through various sensors and are aggregated using some data fusion models. The sensor data serves like back bone to any of surveillance and monitoring systems. The Data Fusion Information Group (DFIG) has proposed a seven layer model for this purpose[9]. The seven stages are Data Assessment, Object Assessment, Situation Assessment, Impact Assessment, Process Refinement, User Refinement, Mission Management.

CHAPTER 2

BACK GROUND

2.1 Undirected Graph

A graph is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges. An undirected graph is one in which edges have no orientation. The edge (a, b) is identical to the edge (b, a), i.e., they are not ordered pairs, but sets {u, v} (or 2-multisets) of vertices. The maximum number of edges in an undirected graph without a self-loop is n(n - 1)/2. In the graph, weights can be assigned to edges of the graph indicating the relationship between the nodes connected by that edge.

A directed graph or digraph is an ordered pair D = (V, A) with V a set whose elements are called vertices or nodes, and A is a set of ordered pairs of vertices, called arcs, directed edges, or arrows. An arc a = (x, y) is considered to be directed from x to y; y is called the head and x is called the tail of the arc; y is said to be a direct successor of x, and x is said to be a direct predecessor of y. If a path leads from x to y, then y is said to

be a successor of x and reachable from x, and x is said to be a predecessor of y. The arc (y, x) is called the arc (x, y) inverted.

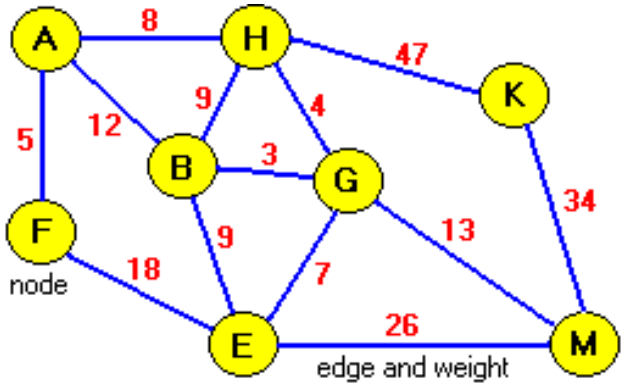

Figure 5.1: Undirected Graph.

2.2 Minimum Spanning Tree

For a given connected graph that is undirected, a spanning tree is a tree that connects each and every node of the graph. In the graph, weights can be assigned to edges of the graph indicating the relationship between the nodes connected by that edge. Sum of all the weights of a spanning tree gives its total weight. Among all the spanning trees a graph can have, the spanning tree that has least weight is called the minimum spanning tree for that graph.

Minimum spanning tree is used in the network design. For example you have a business with several offices. You want to lease them up to connect with each other. As the investment cost depends on the length of cables used in this process, we can use the minimum spanning tree method to connect the offices: the offices represent nodes of the

graphs and the distance between the offices represent edges. Thus the cost can be minimized using the minimum spanning tree method.

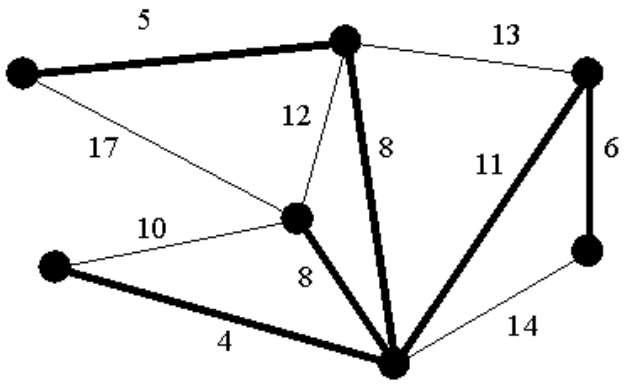

Figure 2.2: Minimum Spanning Tree.

The edges with dark lines in the figure 2.2 above forms the minimum spanning tree.

2.3 Leader Node

In a graph, the leader node is considered as the organizer that coordinates the activity of the other nodes in the graph. Leader node concept need not be applicable to all graphs. But for the graphs with leader nodes, they are the task leaders. In the distributed environment, many algorithms can be executed in order to elect the organizer or task leader for rest of the nodes. In our thesis requirement we assign some vertex as leader node in the input files being passed to compare the graphs. In our thesis when comparing two graphs with leader nodes, along with checking if the same set of edges in both their

minimum spanning trees are having similarity in the common point, it should be checked if the leader nodes are formed by same set of edges in both the trees.

For example, let us consider the following two trees.

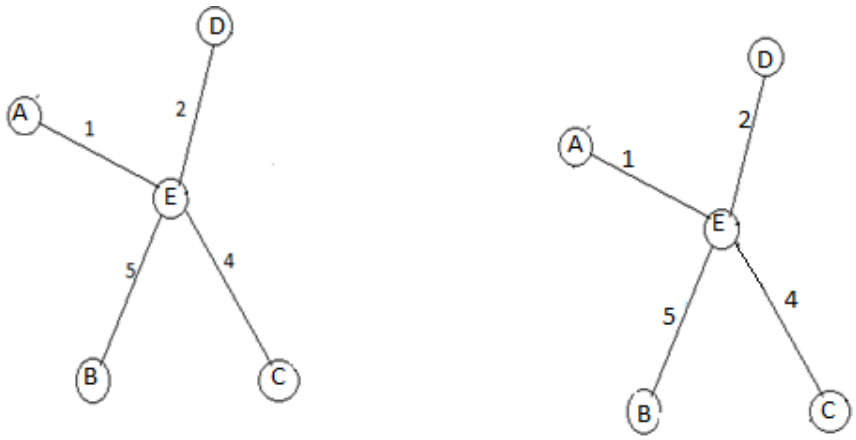

Figure 2.3: Leader Node.

If the leader node in first tree is A and the leader node in the second tree is D, though the trees look similar they are not similar. If the leader nodes are also same, only then we can conclude that the graphs are similar.

2.4 Kruskal's Algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm that finds an edge of the least possible weight that connects any two trees in the forest[24]. It is a greedy algorithm as it finds a minimum spanning tree for a connected weighted graph at each step. This means it finds a subset of the edges that forms a tree that includes every vertex,

where the total weight of all the edges in the tree is minimized. We use Kruskal's algorithm by first sorting the edges and then proceed to other edges in order to eliminate edges that form cycle with the previous edges.

In this algorithm the edge with least weight is part of the minimum spanning tree and from then, each edge is processed to see if that edge can cover other vertices and not forming a cycle. The edges being processed first has more priority over the edges processed later as they have least weight as the edges are arranged in the ascending order.

The complexity of Kruskal's algorithm is **m log n** or **m log m**

where m is number of edges in the graph and

n is the number of nodes in the graph.

m log m is almost equivalent to m log n as a graph can have at most n^2 edges.

m log m = m log n^2

m log m = 2m log n (equivalent to m log n).

2.5 Comparing Two Nonrooted Unoriented Minimum Spanning Trees

The two minimum spanning trees that are being compared are unoriented and not rooted, that is we do not have to take care about the leader node. The first criteria that should be satisfied in order for the comparison to move forward is that, there should be equal no of nodes in both the trees and should have same set of weights of edges. The

comparison says the trees are similar depending on the same condition that is, existence of common vertex or not between any selected pair of edges in both the trees. If this condition holds good we can conclude that the trees being compared are similar.

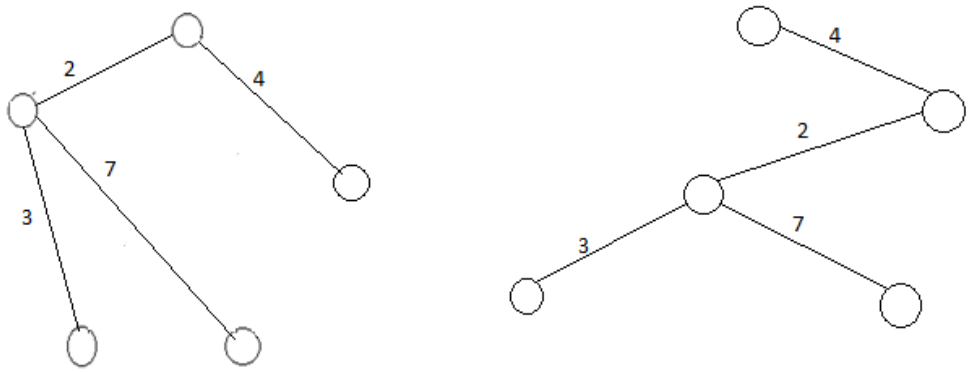

Figure 2.4: Comparing two non rooted trees.

In the Figure 2.4 two trees being compared, they are similar as they follow the rule of availability of common point for all the set of edges. As they are non rooted, we need not check the leader node similarity also.

2.6 Comparing Two Rooted Unoriented Minimum Spanning Trees

The two minimum spanning trees that are being compared here are also unoriented but rooted, that is we should consider the equality at leader node also. This comparison also follows the same initial criteria and the further comparison process is same. In addition to it there should be another check for the similarity at leader node. For

both the trees the leader node should be from the same set of edges. Only then we can say that rooted minimum spanning trees are similar.

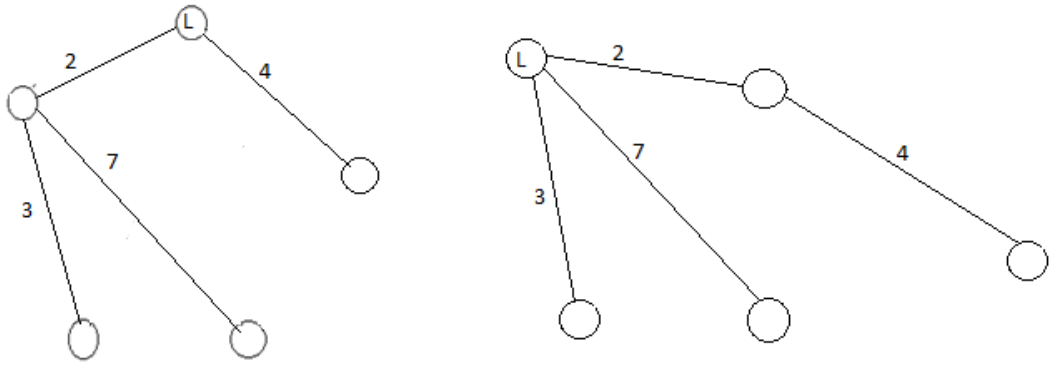

Figure 2.5: Comparing two rooted trees.

In the above two trees that are rooted at L and unoriented, these trees are similar, if we do not consider the leader node. But the leader nodes in the first tree is from the intersection of edges of length 2 and 4, in the second one the leader node is at the intersection of edges of length 2,3, and 7. So these two trees are not similar.

2.7 Situational Assessment

Identifying objects that are participants in a situation and the relationship between these objects. Model situational assessment as a undirected graph in which the objects are the nodes and the relationships are the edges. Because the situational assessment may not

be completely discovered, the graph itself may have missing edges. So when comparing two situations we do not compute the isomorphism of the two graphs modeling the situations respectively, but we compare the minimum spanning trees of the two graphs. That way even if sometimes the graphs are not isomorphic the two situations are deemed similar due to the fact that the minimum spanning trees are the same.

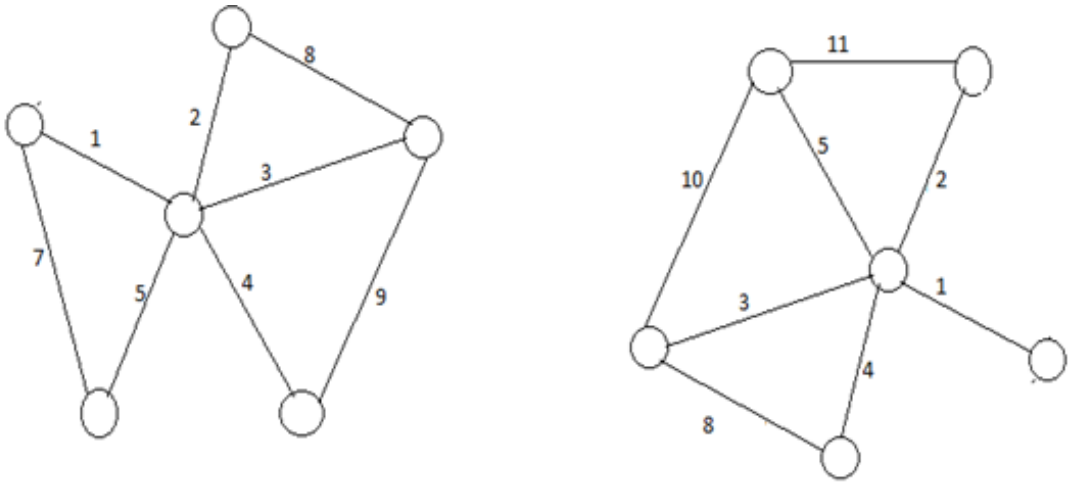

Figure 2.6: Situational Assessment.

In the figure 2.6, there are two graphs which are different from each other. But the similarity between these two graphs is, their minimum spanning trees are made with the same set of edges. The situations are similar, even if the edges that are not needed are removed or new edges that do not alter the spanning tree are added.

In the Data Fusion Information Group's seven layer model for data fusion, the Situational Assessment layer objectives to predict and estimate the relations among

entities that are identified at the objective level [9] where the fusion of information first occurs and it is prior to the Situational Assessment layer. The predictions are used for scene analysis and understanding.

2.8 Edit Distance

Edit distance is used to find the dissimilarity between the trees [3]. The three edit operations that are considered are relabeling, insert and delete operations of the nodes. In relabeling, the label of the node is changed. If a node is deleted, its children are connected to the parent node of deleted node. If a node n is inserted as child of m, n may be parent of the consecutive subsequence of the current children of m. For simplicity the cost of each of these edit operations are considered one. So the minimum sequence of the total no of operations required to convert one tree structure to other becomes the edit distance from one tree to the other. Finding the graph edit distance and find the maximum common subgraph are related. Under some particular cost functions, they are computationally equivalent [23].

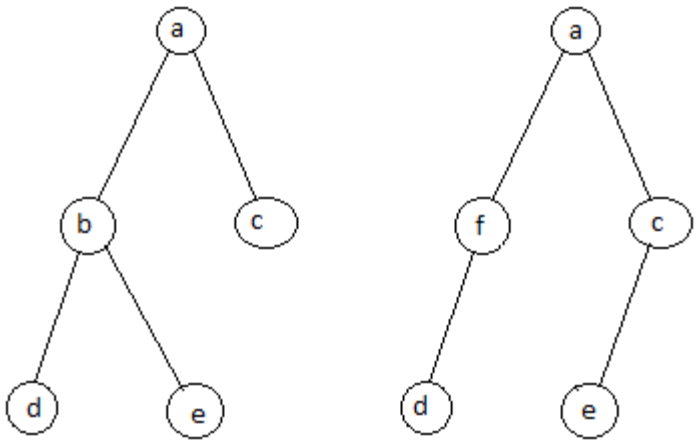

Figure 2.7: Edit distance.

In the figure 2.7 all the edit operations were performed. Node b is relabeled to f. Node e is deleted from node b and inserted to node c. So the edit distance is 3.

CHAPTER 3

ALGORITHMIC APPROACH

3.1 Algorithm to Find the Minimum Spanning Tree

In the situational assessment, we model the situation as a graph in which the objects are considered as nodes and the relationships between them as weighted edges. By finding the minimum spanning tree of the graph, we are considering all the objects in the situation and the crucial relationships consume least amount of effort. There are many algorithms to find the minimum spanning tree with various complexity. But we are using Kruskal's algorithm to find the minimum spanning tree for the given edges and vertices because of its time complexity which is m log n where m is no of edges and n is no of vertices of the graph.

The detailed steps of the algorithm are

- Arrange the edges of the given graph in increasing order of their weights.
- Consider the least weighted edge as part of the minimum spanning tree.

- As we progress by the increased order of the edges, compare the parental vertices of both the end points of the edge. Parental vertex of a set of vertices, is the vertex that has maximum edges connected to it.
- If the parental vertices of both the end points of the edge are equal, it means the new edge is forming a cycle and has to be ignored.
- If they are not equal Then include that edge in to minimum spanning tree.
- Once they are added to the minimum spanning tree, the rank of the vertices should be increased by 1 and combine the set of vertices of both the end points of the edge into one set.
- Once all the edges are read, we get the minimum spanning tree of all the edges.
- This a greedy algorithm, as it wants to find the minimum spanning tree at each level of edge being read.

```
360    public static int Find (int vertex){
361        if(PARENT.get(vertex)==vertex)
362            return PARENT.get(vertex);
363
364        else
365            return Find(PARENT.get(vertex));
366    }
367    public static void Union(int root1, int root2) {
368        int num;
369        num=RANK.get(root2);
370        if( RANK.get(root1) > RANK.get(root2))
371            PARENT.put(root1, root2);
372        else if(RANK.get(root1) < RANK.get(root2))
373            PARENT.put(root2, root1);
374        else
375            PARENT.put(root2, root1);RANK.put(root2,num++);
376    }
377
378    public static Edge[] minTree(Edge[] ed,int v, int e) {
379        //Sorting edges
380        for(int i=0;i<e;i++)
381        for(int j=0;j<e-1;j++)
382        {
383            if(ed[j].wt>ed[j+1].wt)
384             {
385                Edge t=new Edge();
386                t=ed[j];
387                ed[j]=ed[j+1];
388                ed[j+1]=t;
389            }
390        }
391
392        for(int i=0;i<v;i++) {
```

```
387                ed[j]=ed[j+1];
388                ed[j+1]=t;
389             }
390      }
391
392      for(int i=0;i<v;i++) {
393          int x=Integer.parseInt(String.valueOf(SetVertices.get(i)).substring(0,1));
394          PARENT.put(x,x);
395          RANK.put(x,0);
396      }
397
398      for (int i=0;i<e;i++) {
399          int root1=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v1.a)).substring(0,1)));
400          int root2=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v2.a)).substring(0,1)));
401          if(root1!=root2){
402              ed[i].con=1;
403              Union(root1,root2);
404          }
405      }
406          return ed;
407          }
408
```

3.2 Algorithm to Compare Two Minimum Spanning Trees Without a Leader Node

After finding the minimum spanning tree of the situation before and after a mission plan, we now have to find the differences between these two situations, which gives the effect of the mission plan executed. To compare the two situations, we have the corresponding minimum spanning trees to compare. The algorithm checks for the equality condition with respect to the existence of the common point between the same set of edges in both the spanning trees. This algorithm is to compare the tress that do not have leader node. The detailed steps of the algorithm to compare two trees without a leader node is the following

- Compare the no of nodes of each graph and exit if not equal.
- Generate minimum spanning trees for both A and B graphs.
- Add the weights of each min spanning tree separately and compare the sum. Exit if not equal.
- In the generation of minimum spanning tree the weights are already arranged in increasing order. So check if the individual weights in the spanning tree of one graph is matching with the other graph and exit if not equal.
- If it is matching, check if the pair of weights in one graph has a vertex in common or not and the same should be the result for the same pair in the other graph.
- This check should be done for all the possible pairs of both the graphs.
- If it violates for at least one pair, we can declare that the graphs are not similar.
- If all the pairs have same condition in both the graphs, then we can conclude that the graphs are similar.

```
        if(v!=v2){
            check=1;
        }
        else if(w1!=w2) {
            check=1;
        }
        else {
            for (int i=0;i<v-1;i++)
            {
                if(min_ed[i].wt!=min_ed2[i].wt)
                {
                    check=1;
                    break;
                }
            }
        }

        if (check==0)
        {
            for (int i=0;i<v-1;i++)
            {
                for(int j=i+1;j<v-1;j++)
                {
                    if(min_ed[i].v1.a==min_ed[j].v1.a || min_ed[i].v1.a==min_ed[j].v2.a ||
                            min_ed[i].v2.a==min_ed[j].v1.a || min_ed[i].v2.a==min_ed[j].v2.a)
                        common_chk1=1;
                    if(min_ed2[i].v1.a==min_ed2[j].v1.a || min_ed2[i].v1.a==min_ed2[j].v2.a ||
                            min_ed2[i].v2.a==min_ed2[j].v1.a || min_ed2[i].v2.a==min_ed2[j].v2.a)
                        common_chk2=1;
                    else
                        common_chk2=0;
                    if(common_chk1!=common_chk2){
                        check=1;
                        break;
```

3.3 Algorithm to Compare Two Minimum Spanning Trees with Leader Nodes

The comparison between two graphs with a leader node follows a different approach. Along with the checking the similarity in existence of the common points, the leader node should also be checked in both the graphs. If both these conditions are valid, then the minimum spanning trees can be considered similar. The algorithm to find the similarity of the minimum spanning trees with leader node is following

- In this type of problem the leader node in the graph should also be provided as part of input.
- The initial criteria of matching the no of nodes should be valid to proceed forward.
- Finding the minimum spanning trees follows the same algorithm as stated above.
- Add the weights of each min spanning tree separately and compare the sum. Exit if not equal.
- In the generation of minimum spanning tree the weights are already arranged in increasing order. So check if the individual weights in the spanning tree of one graph is matching with the other graph and exit if not equal.
- Later the leader nodes are to be passed to the function that determines the similarity of the graphs.
- The similarity function checks if the input arguments given to it are leaf nodes.
- If yes then it returns true.
- If not, then array of edges connected to the nodes should match in both the graphs. if it does not match, exit stating not similar.

- If matched, then the equality function is called recursively with the nodes connected by same weights in both the graphs and equality should hold good all over.
- If the weight comparison of the input arguments fails the function returns false.
- Thus we evaluate the similarity of the graphs with leader node.

```
274    public static boolean check_equal(int a,int b) {
275        System.out.println("start  "+a+ "  "+b);
276        int index1=-1,index2=-1,other1=-1,other2=-1,index=0;
277        boolean result=true;
278        float chk1,chk2;
279        for(int i=0;i<v;i++)
280        {
281            if(ar[0][i]==a && ar[2][i]==0){
282                index1=i;
283                ar[2][i]=1;
284                break;
285            }
286
287        }
288        for(int j=0;j<v;j++)
289        {
290            if(ar2[0][j]==b && ar2[2][j]==0){
291                index2=j;
292                ar2[2][j]=1;
293                break;
294            }
295        }
296        if(index1==-1) {
297            return true;
298        }
299        System.out.println(String.valueOf(SetVertices.get(index1)).substring(0,1));
300        //System.out.println(index2);
301        if(ar[1][index1]==0 && ar2[1][index2]==0)
302            return true;
303        else {
304            for(int i=0;i<v-1;i++)
305            {
306                chk1=-1;chk2=-1;
307                if(Integer.parseInt(String.valueOf(SetVertices.get(min_ed[i].v1.a)).substring(0,1))==a){
```

```
304             for(int i=0;i<v-1;i++)
305             {
306                 chk1=-1;chk2=-1;
307                 if(Integer.parseInt(String.valueOf(SetVertices.get(min_ed[i].v1.a)).substring(0,1))==a){
308                     chk1=min_ed[i].wt;
309                     other1=min_ed[i].v2.a;
310                 }
311                 else if (Integer.parseInt(String.valueOf(SetVertices.get(min_ed[i].v2.a)).substring(0,1))==a){
312                     chk1=min_ed[i].wt;
313                     other1=min_ed[i].v1.a;
314                 }
315                 //System.out.println("1  "+ chk1+"  "+other1);
316                 if(Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[i].v1.a)).substring(0,1))==b){
317                     chk2=min_ed2[i].wt;
318                     other2=min_ed2[i].v2.a;
319                 }
320                 else if (Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[i].v2.a)).substring(0,1))==b){
321                     chk2=min_ed2[i].wt;
322                     other2=min_ed2[i].v1.a;
323                 }
324                 //System.out.println("2  "+ chk2+"  "+other2);
325                 if(chk1!=chk2){
326                     index=1;
327                     break;
328                 }
329                 else if(chk1!=-1) {
330                     if(ar[2][other1]==0 &&
331                             ar2[2][other2]==0)
332                             result=result && check_equal(Integer.parseInt(String.valueOf(SetVertices.get(other1)).substring(0,1
```

```
332                             result=result && check_equal(Integer.parseInt(String.valueOf(SetVertices.get(other1)).subst
333                                     Integer.parseInt(String.valueOf(SetVertices.get(other2)).substring(0,1)));
334                 }
335             }
336             if (index==1)
337                 return false;
338             else
339                 return result;
340         }
341     }
```

CHAPTER 4

IMPLEMENTATION

In this chapter we present the details that are related to the actual coding, the description of the environment and the technology that are related to our thesis.

4.1 Programming Environment and Setup

The coding for this thesis is implemented in Java programming language. Java is a high level programming language. The language's syntax is derived from C and C++ languages. It is platform independent which means the programs written in Java language must run similarly on any combination of operating system and hard ware. It is achieved by an intermediate representation called Java bytecode formed by compiling the Java language. A .java file is compiled into bytecode stored in .class file. The file is then executed by JVM(Java Virtual Machine) in any operating system at the end user. So the programs written and compiled in the Java language are portable. The Java version that is used for coding in our thesis is 1.8.0.

The IDE (Integrated development environment) in which the code related to our thesis is developed is eclipse. The eclipse version that was being used is Luna. JavaFX the new framework for developing Java GUI programs is also used in our thesis. As our thesis is related to the similarity of graphs, a pictorial representation of the differences is needed. For this purpose we used the JavaFX framework.

4.2 Code Structure

The code for two types that is with leader node and without leader node has the following 3 files. The Main program file's code and the input files data only changes in these two types.

- a Class file named Vertex
- a Main program file.
- two input files representing two situations/ graphs.

All the three files are enclosed in a single package. The class file vertex is common in the two types of coding requirements i.e. with leader node and without leader node.

4.2.1 Vertex Class File

The class file vertex is created as a representation of nodes we are going to use in the main file. The two variables that are declared are unknown index and name. The class file has got constructors which gives values to these variables when the object of the class is created. The functions of the class file are to input the value to the variable name or return the value that is stored in the variable name and also to return the index.

4.2.2 Main Program without a Leader Node

4.2.2.1 Class Edge

We used a class called edge which represents the edge in the graph. The attributes of Edge are the vertices which in turn belong to class vertex defined in the class Edge, the weight of that edge and a flag con which describes if the edge is part of its minimum spanning tree.

4.2.2.2 Class MinimumSpanningTree

All the coding that is required for the thesis is written in this class. The JavaFX code that is used to draw the graphs comparing the differences of the graphs being compared. It has a function that takes in the array of edges of a graph as input and returns the array of edges that forms the minimum spanning tree of the graph. It has the main function which will be discussed in detail below.

4.2.2.3 minTree Function

This function in the class MinimumSpanningTree gives the minimum spanning tree of the graph given as input to the function. We used Kruskal's algorithm for find the minimum spanning tree. As Kruskal's algorithm works on the sorted edges, the code for arranging the edges in sorted order is also written here. Every time when a new edge is read, it is checked if it is not forming a cycle. This function returns the array of Edges which represent the set of edges of the graph that forms minimum spanning tree that covers all the vertices of the graph. We also make use of two functions in deriving the minimum spanning tree in this function. We used Map data structure for storing the vertices and the rank of the vertices.

- **Find()**

  Find function is used to find the parental vertex of the vertex passed as the parameter to the function. Parental vertex is the vertex that has maximum degree (no of edges from the vertex) of the vertices connected till then. If a vertex is not connected to any of the edge, the parental vertex of that vertex is itself. We used hash maps to store these details.

- **Union()**

  Union function is used to join the set of vertices of both the endpoints of the edge being read if their parental vertices are not same. This conveys that the edge is part of minimum spanning tree.

Before calling the functions described above, each vertex is made parent of itself and rank of each vertex is initialized to 0. It is the rank value that determines which vertex is parent among the set of vertices connected till then.

4.2.2.4 Main Function

In this function the code to accept the input files and format them as per the variables in the classes described above the main function. Once the formatting is done, the minTree function is called with the array of edges as parameters to the function. Once the processing for first input file is completed, the same process is applied for the second graph. So we have two arrays of edges which represent the minimum spanning trees of the two graphs being compared. The coordinates of vertices from the input files are also stored in the array list and passed to the JavaFX code for marking the vertices on the graph. As per the algorithm described above, we check the common point availability of the same set of edges in both the graphs and they should match for all the set of edges. Once all the edges are completed, the flag is updated to display if the graphs are similar or not and the same will be displayed in the graphs that will be populated using the JavaFX code.

4.2.3 Main Program File with Leader Node

The main program file with leader node has some similar functionality when compared with the main program file without a leader node. It uses the same class file Edge which describes the edge of a graph with same attributes in the class file. The minTree function that takes in the input as array of edges of the graph and returns the minimum spanning tree edges as output. Even in the main program file without the leader node, we use the minimum spanning tree of the graphs and find the similarity between the graphs being compared.

4.2.3.1 check_equal Function

This function is new when compared with the program file without a leader node. This function is defined as a recursive function. This function returns the Boolean value i.e. if the vertices that are passed as input and being compared are similar or not. If they are similar, it returns true and false if otherwise. This function is initiated by passing the leader nodes of both the graphs. This function checks if the edges of the minimum spanning tree that are connected to the vertices passed as input are common in both the graphs. If the vertices passed are leave nodes it returns true. If the array of weights that are connected to vertices are same, then the check_equal function is called correspondingly with the other vertices of the edges matched. If at any point the weights do not match, the function returns false stating the graphs are dissimilar.

4.2.3.2 Main Function

In the main function we have code to access the input files and also do the formatting of the to match the variables defined in the classes and functions defined previously. It initially calls the minTree function in order to get the minimum spanning trees of the graphs. The co ordinates of the vertices are also stored for creating the graph using JavaFX code. Once the initial process is done the later code follows the procedure described in the algorithms above. It calls the check_equal functions with leader nodes of both the graphs and the function is recursively executed and gives the result of the similarity of the graphs.

4.3 Input Files

We have two input files related to the two graphs being compared without leader node. Both the files have same format of data but the values will be different. The input file has initially the number of vertices of the graph. It is followed by the number of the edges of the graph and then the labels of the vertices. Then we have the details of the edges of the graph. Depending on the order of labels of vertices, we use indices of the vertices between which the edge is and its weight as one per row. after all the edges are read, the coordinates of the vertices are given as ordered pairs in the same order as the labels were given initially. This input file format is to compare graphs that do not have leader nodes.

For the input files that are given for comparing the graphs with leader nodes is almost same except for the indication of the label of the leader node. The leader node label is given after the number of edges and before the labels of the vertices. All the values in the input files for with or without leader nodes are arranged one per each line. The labels of the vertices can be alphabets but when describing the edges after the labels of the vertices, use numbers as they are indices.

CHAPTER 5

RESULTS

This chapter describes the results that we have obtained for various types of inputs representing graphs with leader nodes and without leader nodes. We have implemented the code ion Java using Eclipse as an IDE.

5.1 Comparing Two Trees without Leader Node

In the graphs that are shown below there are few points that should be considered.

- The graphical output displayed has two panes i.e. two graphs one on top of the other representing the graphs being compared.
- The entire graph will be displayed i.e. it has edges that belong to the minimum spanning tree and edges that do not belong to minimum spanning tree.
- The edges that belong to the minimum spanning tree are highlighted in blue.
- The width of each edge is designed such that it is directly proportional to the weight of the edge.

- The title of the graphical popup window shows the result if the graphs are similar or different.

5.1.1 Two Graphs are Same

If the graphs that are being compared are the same, meaning that that they have same number of vertices and the same weight for the set of edges, then the graphs are deemed similar per our algorithm.

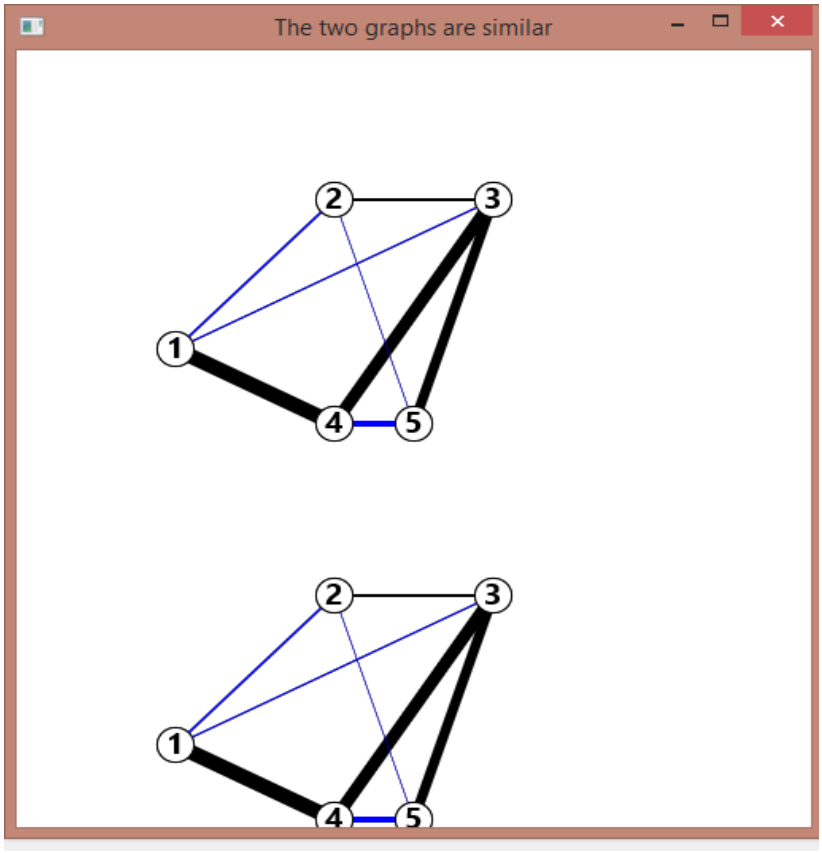

Figure 5.1: Result of comparing two same graphs.

The edit distance for the two trees in the figure 5.1 is 0. As the graphs are similar concluded by the same minimum spanning trees of the graphs, there is no need of additional deletions or insertions of the edges required.

5.1.2 Two Graphs with Same Minimum Spanning Trees

If two graphs are different but the minimum spanning trees are the same (i.e. the minimum spanning trees have the same number of vertices and the same weight for the set of edges), then we consider the graphs to be similar per our algorithm.

The edit distance for the two trees shown in the figure 5.2 is 0. As no additional insertions or deletions or relabeling is not required. So when the graphs are similar, which will be concluded by similarity in minimum spanning trees their edit distance is zero.

In the figure 5.2, the edge from 1 to 4 is not present in second graph, as it is not part of minimum spanning tree , it does not make any difference and concludes that the graphs are similar.

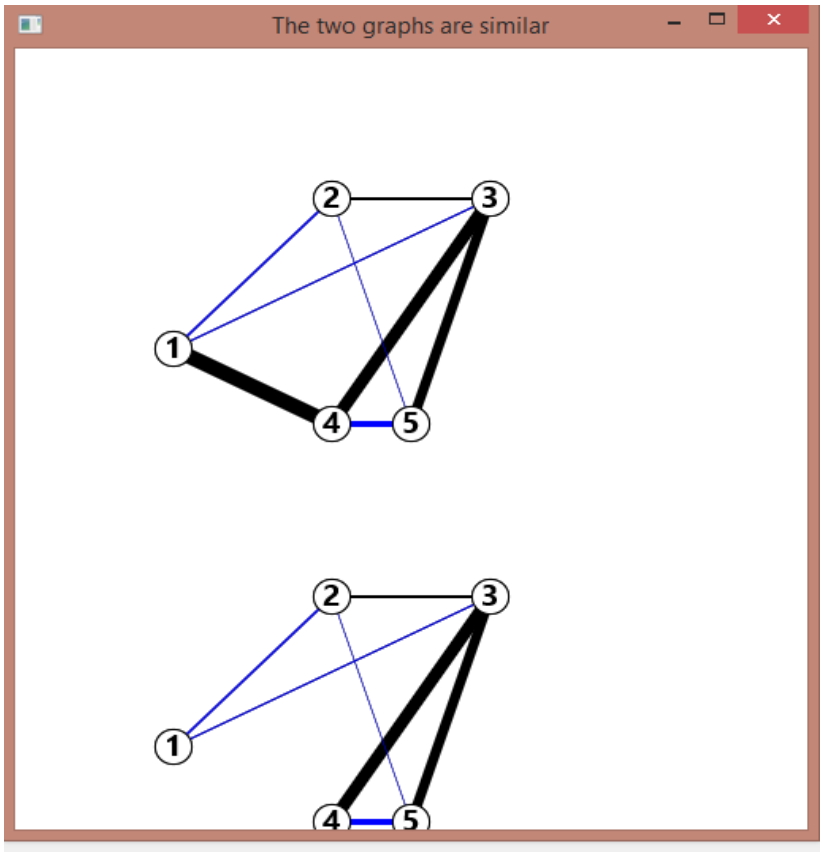

Figure 5.2: Results for comparing two graphs with same minimum spanning tree.

5.1.3 Two Graphs with Different Minimum Spanning Trees

This case is different from the previous one: the graphs are different and their minimum spanning trees are also different. Then our graph comparison algorithm concludes that they are not similar.

In the pop up window in figure 5.3, an edge from 4 to 5 is missing in second graph, which is part of minimum spanning tree. So the graphs are deemed not to be similar by our algorithm.

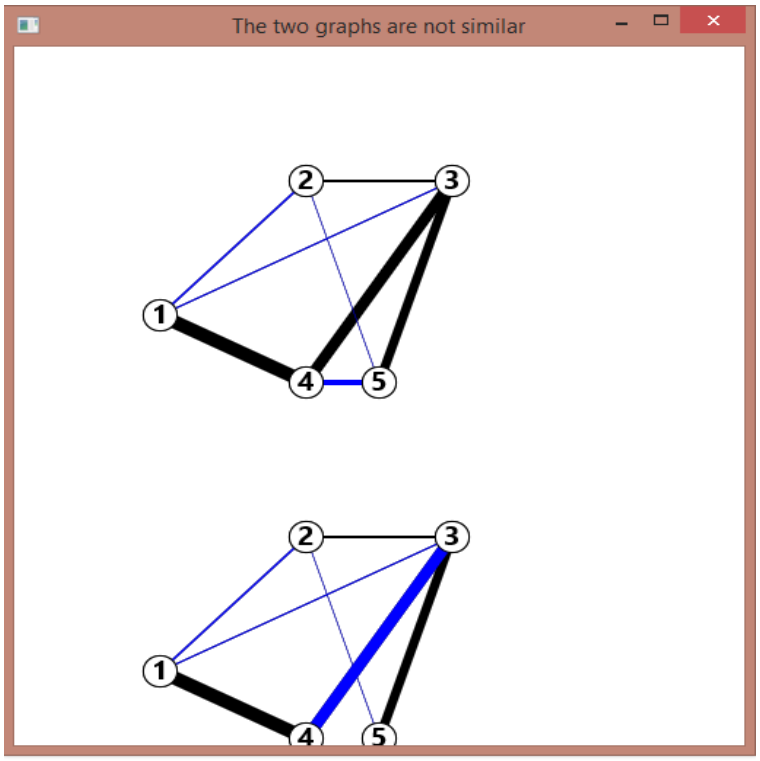

Figure 5.3: Results of comparing two graphs with different minimum spanning trees.

The edit distance for the figure 5.3 is two as only one delete operation and one insertion operations are required on first graph to form the second graph. The node 4 is removed from its parent 5 and inserted as child to node 3. This process involved two edit operations and hence the edit distance is 2.

5.2 Comparing Two Trees With Leader Nodes

The comparison of the minimum spanning trees with leader nodes follows a different approach at coding level. In the graphical display of the results, there are few points that are to be considered.

- In the graphs the leader node is highlighted in red indicating the importance of that node as compared to the others.
- The edges of the minimum spanning tree are highlighted in blue.
- The width of edges are designed such that they are directly proportional to the weight of the edges.
- The title of the popup window shows whether the graphs being compared are similar or different.

5.2.1 Two Graphs with Different Minimum Spanning Trees

If the two graphs that are being compared are different and they have different minimum spanning trees then our algorithm concludes that the graphs are not similar.

For example, let us consider the graphs in the figure 5.4. The first graph is different from the second graph. The minimum spanning tree that is highlighted in blue in both the graphs are also different. The edge from 2 to 3 that is part of minimum spanning tree in the first graph is missing in the second graph. The edge from 2 to 4 that is part of minimum spanning tree in the first graph is also missing in the second graph. Thus the minimum spanning tree in second graph is longer than that of minimum spanning tree of the first graph.

We are comparing the situations before and after a mission plan is executed. As the minimum spanning tree varied in the second graph indicates that the mission plan executed has effect on the situation.

The edit distance for the figure shown 5.4 is four. As the minimum spanning trees of both the graphs varied, there will be a positive value for the edit distance. In this example the nodes 4 and 3 are deleted from their parent 2. The node 4 is inserted as child to node 5 and node 3 as child to node 1. This operation involves two insert operations and two delete operations making the edit distance as four.

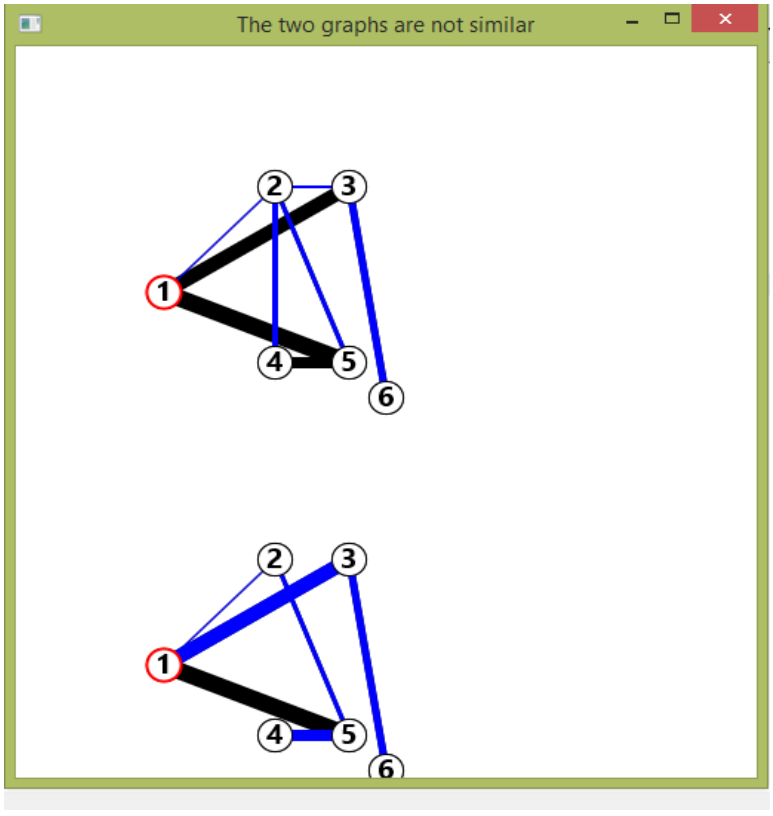

Figure 5.4: Results of comparing two graphs with leader nodes with different minimum spanning trees.

5.2.2 Two Same Graphs

If the two graphs that are being compared are the same i.e. the nodes and the weight of edges among these nodes are the same, our algorithm concludes that the graphs are similar and this is reflected into the title of the popup window.

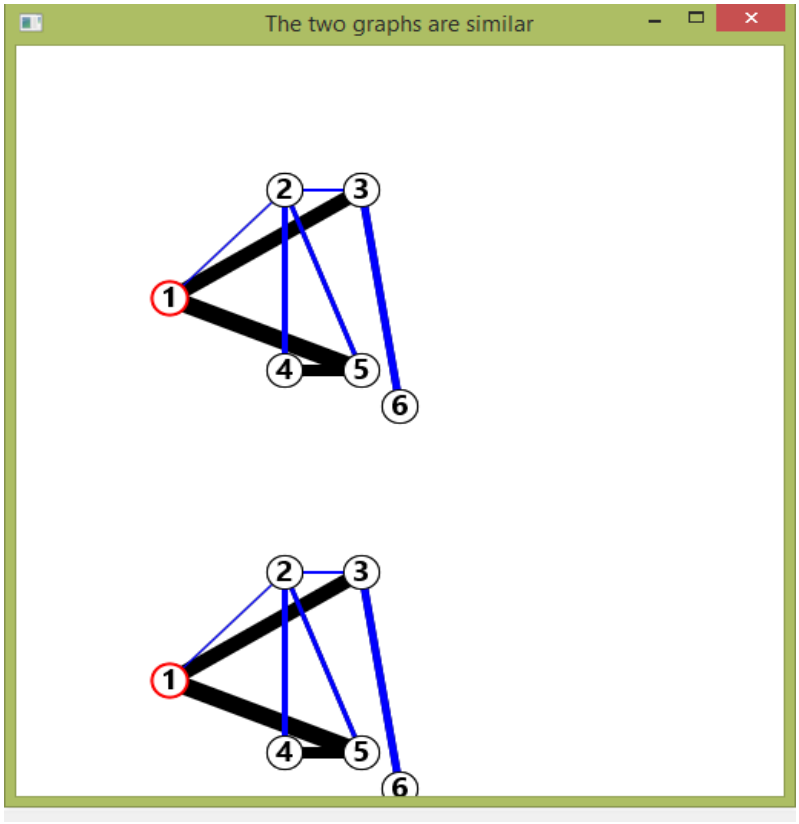

Figure 5.5: Results of comparing two same graphs with leader nodes

The edit distance for the figure 5.5 is zero. As the graphs are same and have same minimum spanning trees, there is no need of additional edit operations making the edit distance zero.

5.2.3 Two Graphs with Same Minimum Spanning Trees

If the two graphs that are being compared are different but the set of edges that makes the minimum spanning tree and the weight of those edges are the same, then our algorithm concludes that they are similar even if they are different.

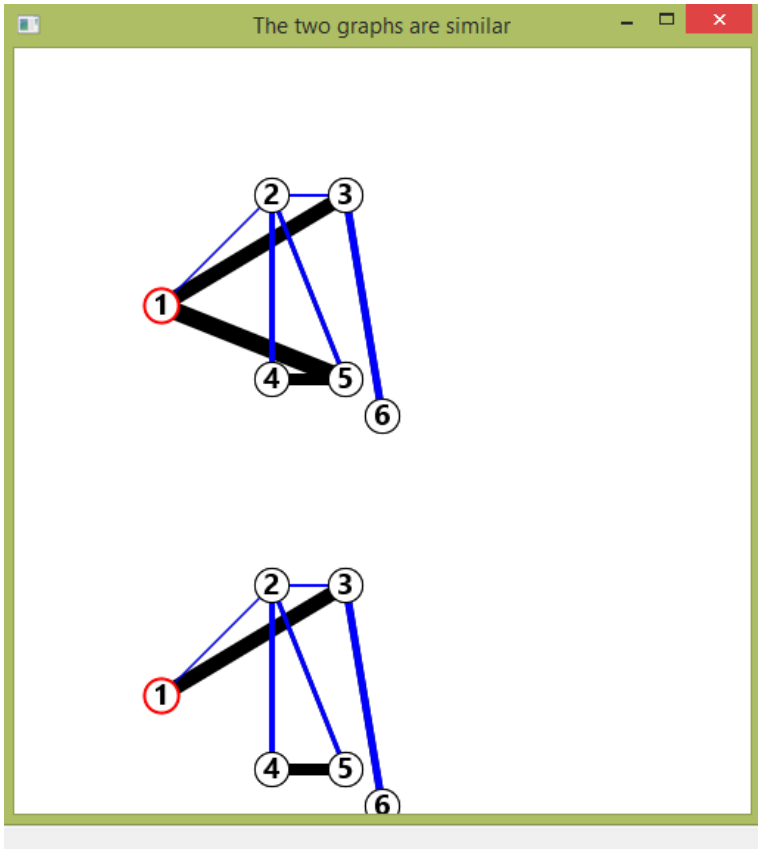

Figure 5.6: Results of comparing two graphs with leader nodes with same minimum spanning trees.

For example in the figure 5.6, the edge from 1 to 5 is missing which is not part of minimum spanning tree as it is not highlighted in blue. That edge is missing in the

second graph. We are concerned only with the edges of the minimum spanning tree and hence the graphs are similar.

The edit distance in the figure 5.6 is zero as the minimum spanning trees are same, no additional operations are required.

CHAPTER 6

DOCUMENTATION

6.1 Configuring and Running the Project

In this section we describe how we have configured the project. The project name is Rooted_Spanning. The coding for the project is done in Java. The integrated development environment (IDE) is eclipse and the version we used is Luna. As we need JavaFX for display of the graphs, we should ensure that appropriate packages also should be installed. After opening eclipse, to open the project, click the following menus in the sequence: File > New > Java Project. This opens the new Java project dialog box. Uncheck the use default location checkbox and browse for the location of the project. Once it is done, click on the Finish button to open the project. See the figure 6.1 for hints.

Once the project is brought in to the work space, we should select the Run option from the Run menu. We can also press the Run icon present on the window. On a windows operating system, we can use ctrl + F11 to run the file. The sample input files required to run the project are already available in the package. By following the format,

the data can be altered to monitor different results. The description and format of the input files that should be used are briefly mentioned in 4.3.

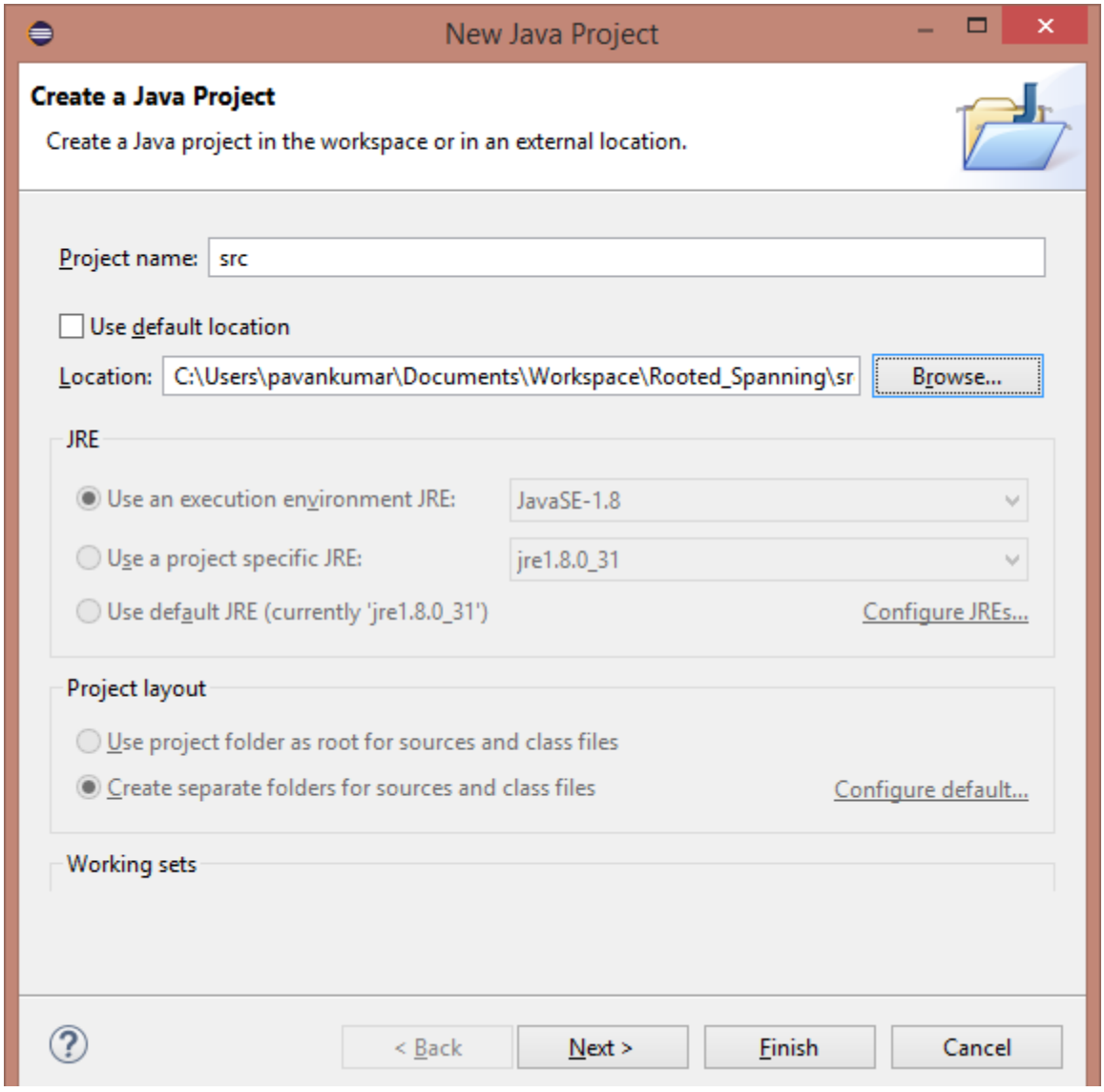

Figure 6.1: Screen shot of configuring the project

6.2 Code Structure and Flow

The execution of the first project starts at the main() and reads the data from the input files and stores in the arrays. These arrays are passed to minTree() for finding the minimum spanning tree of the data from input files. The minimum spanning trees are used to compare for the existence of common points between edges in the main(). The results and the input values are passed to the JavaFX coding part in the class file and it displays the graph with result as title of the pop up window.

In the second project with leader nodes, the execution till finding the minimum spanning trees is same. Then the leader nodes are passed to check_equal() which is a recursive function and gives the result about the similarity of the graphs. The results, coordinates and minimum spanning tree edges are passed to JavaFX code for displaying the graph.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we modeled a situation like battle field in to a graph with nodes and edges. The objects in the battle field are the nodes and relationships between them are the edges of the graph. By proposing an algorithm to find the similarity between the same situation at different intervals of time, we are able to determine the effect of the a mission plan executed. For designing an effective mission plan, this analysis of the previous mission plan is very important. Thus helping in successful designing of the mission plan.

This approach is applicable to the situation with a leader or without a leader. In scenarios like battle field, the object that is organizer and controller of other objects in the same field are considered as the leader and are given more priority in determining the effect of the mission plan. For example if there is a central hub that communicates information to other objects and controls them, if the mission plan is able to effect that central hub the loss is more concluding that the mission plan was every effective.

By using the concept of minimum spanning tree, we can reduce the calculations and complexity in comparing the situations at two different points of time to a very large

extent. Thus reducing the time taken for analyzing the data of the situation on which mission plan is going to be executed. For military mission plans, time is an important factor as the probability of situation to change increases with the increase of time. Thus effective mission plans can be made by reducing the time taken to analyze the data.

The enhancement that can be done to the thesis work is to give a measure like a degree of difference or the amount of difference between the two minimum spanning trees being compared. That helps in quantifying the effect of the mission plan implemented on a situation at various time intervals.

The thesis work can also be enhanced to compare the similarity of the situations if the relationships between objects in the situation are have one way communication. Such situations can be modeled as directional graphs in which the edges between vertices can be one directional.

APPENDIX A

CODE SNIPPET FOR THE CLASS VERTEX

```
package MainPackage;


/**
 * A class for vertices in graphs.  Every vertex has a name and an
 * index in its graph.
 *
 */

public class Vertex
{
   /** Value that indicates that this vertex does not yet have an
    * index, i.e., the index is unknown. */
   public static final int UNKNOWN_INDEX = -1;

   /** Index of this vertex in its graph, 0 to cardV-1. */
   private int index;

   /** This vertex's name. */
   private String name;

   /**
    * Creates a vertex whose index is unknown.
    *
    * @param name This vertex's name.
    */
   public Vertex(String name)
   {
         index = UNKNOWN_INDEX;
         this.name = name;
```

```
}

/** Creates a vertex with a given index and name.
 *
 * @param index This vertex's index.
 * @param name This vertex's name.
 */
public Vertex(int index, String name)
{
	this.index = index;
	this.name = name;
}

/**
 * Sets this vertex's index.
 *
 * @param index New value for this vertex's index.
 */
public void setIndex(int index)
{
	this.index = index;
}

/** Returns this vertex's index. */
public int getIndex()
{
	return index;
}

/**
 * Sets this vertex's name.
 *
 * @param name New value for this vertex's name.
 */
public void setName(String name)
{
	this.name = name;
}

/** Returns this vertex's name. */
public String getName()
{
	return name;
}

/** Returns the <code>String</code> representation of this
```

```
     * vertex. */
    public String toString()
    {
          return name + " (index = " + index + ")";
    }
}
```

APPENDIX B

CODE SNIPPET FOR COMPARING TWO TREES WITHOUT A LEADER NODE

```
package MainPackage;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import  java.util.Map;
import java.util.HashMap;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

class Edge
{
        class vertex
        {
                int a;
        }
         vertex v1;
```

```
        vertex v2;
        float wt;
        int con;
        Edge()
        {
                v1=new vertex();
               v2=new vertex();
        }
}

public class MinimumSpanningTree extends Application {
        static List<java.awt.Point> gr;
        static List<java.awt.Point> gr2;
        static Edge min_ed[]=new Edge[200];
        static Edge min_ed2[]=new Edge[200];
        static Edge ed[]=new Edge[200];
        static Edge ed2[]=new Edge[200];
        static Map<Integer, Integer> PARENT= new HashMap();
        static Map<Integer, Integer> RANK= new HashMap();
        //create the set of vertices
        static List<Vertex> SetVertices = new ArrayList<Vertex>();
        static List<Vertex> SetVertices2 = new ArrayList<Vertex>();
        private Pane pane;
        private Pane pane2;
        private VBox vertBox;
        static int v,e,v2,e2,check=0;
        @Override // Override the start method in the Application class
        public void start(Stage primaryStage) {
        // Create a pane to hold the graph
        pane = new Pane();
        pane2=new Pane();
        vertBox = new VBox(10);
        // retrieve the number of nodes
        int totalVertices = gr.size();
        int totalVertices2 = gr2.size();
        // define the array of nodes as colored circles
        Circle circ[] = new Circle[100];
        Circle circ2[] = new Circle[100];
        // define and draw each node in the graph
        java.awt.Point node1;
        for(int k=0; k<totalVertices; k++)
        {
                node1 = gr.get(k);
                // define a generic circle for each node
                circ[k] = new Circle();
                circ[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
```

```
            circ[k].setStroke(Color.BLACK);
            circ[k].setFill(Color.BLACK);
            circ[k].setCenterX(node1.x);
            circ[k].setCenterY(node1.y);
            // Add the circle to the pane
            pane.getChildren().add(circ[k]);
}

java.awt.Point node2;
for(int k=0; k<totalVertices2; k++)
{
            node2 = gr2.get(k);
            // define a generic circle for each node
            circ2[k] = new Circle();
            circ2[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
            circ2[k].setStroke(Color.BLACK);
            circ2[k].setFill(Color.BLACK);
            circ2[k].setCenterX(node2.x);
            circ2[k].setCenterY(node2.y);
            // Add the circle to the pane
            pane2.getChildren().add(circ2[k]);
}

//draw the graph

Line line2 = new Line();
for(int k=0; k<e; k++)
{
            // create the arc between node k and its adjacent nodes
            float w=ed[k].wt;
            int n=ed[k].v1.a;
            int l=ed[k].v2.a;
            line2 = new Line();
            line2.startXProperty().bind(circ[n].centerXProperty());
            line2.startYProperty().bind(circ[n].centerYProperty());
            line2.endXProperty().bind(circ[l].centerXProperty());
            line2.endYProperty().bind(circ[l].centerYProperty());
            line2.setStrokeWidth(w);
            line2.setStroke(Color.BLACK);
            // Add the arc to the pane
            pane.getChildren().add(line2);
}

Line line22 = new Line();
for(int k=0; k<e2; k++)
{
```

```
        // create the arc between node k and its adjacent nodes
        float w=ed2[k].wt;
        int n=ed2[k].v1.a;
        int l=ed2[k].v2.a;
        line22 = new Line();
        line22.startXProperty().bind(circ2[n].centerXProperty());
        line22.startYProperty().bind(circ2[n].centerYProperty());
        line22.endXProperty().bind(circ2[l].centerXProperty());
        line22.endYProperty().bind(circ2[l].centerYProperty());
        line22.setStrokeWidth(w);
        line22.setStroke(Color.BLACK);
        // Add the arc to the pane
        pane2.getChildren().add(line22);

}


// draw the edges in the graph
Line line1 = new Line();
for(int k=0; k<totalVertices-1; k++)
{
        // create the arc between node k and its adjacent nodes
        float w=min_ed[k].wt;
        int n=min_ed[k].v1.a;
        int l=min_ed[k].v2.a;
        line1 = new Line();
        line1.startXProperty().bind(circ[n].centerXProperty());
        line1.startYProperty().bind(circ[n].centerYProperty());
        line1.endXProperty().bind(circ[l].centerXProperty());
        line1.endYProperty().bind(circ[l].centerYProperty());
        line1.setStrokeWidth(w);
        line1.setStroke(Color.BLUE);
        // Add the arc to the pane
        pane.getChildren().add(line1);
}

Line line12 = new Line();
for(int k=0; k<totalVertices2-1; k++)
{
        // create the arc between node k and its adjacent nodes
        float w=min_ed2[k].wt;
        int n=min_ed2[k].v1.a;
        int l=min_ed2[k].v2.a;
        line12 = new Line();
        line12.startXProperty().bind(circ2[n].centerXProperty());
        line12.startYProperty().bind(circ2[n].centerYProperty());
```

```
            line12.endXProperty().bind(circ2[l].centerXProperty());
            line12.endYProperty().bind(circ2[l].centerYProperty());
            line12.setStrokeWidth(w);
            line12.setStroke(Color.BLUE);
            // Add the arc to the pane
            pane2.getChildren().add(line12);
}

// re-draw the nodes in the graph to overlap over the edges
for(int k=0; k<totalVertices; k++){
            node1 = gr.get(k);
            // re-define a generic circle for each node
            circ[k] = new Circle();
            circ[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
            circ[k].setStroke(Color.BLACK);
            circ[k].setFill(Color.WHITE);
            circ[k].setCenterX(node1.x);
            circ[k].setCenterY(node1.y);
            // Add the circle to the pane
            pane.getChildren().add(circ[k]);
            }

for(int k=0; k<totalVertices2; k++){
            node2 = gr2.get(k);
            // re-define a generic circle for each node
            circ2[k] = new Circle();
            circ2[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
            circ2[k].setStroke(Color.BLACK);
            circ2[k].setFill(Color.WHITE);
            circ2[k].setCenterX(node2.x);
            circ2[k].setCenterY(node2.y);
            // Add the circle to the pane
            pane2.getChildren().add(circ2[k]);
            }

// Read and display the labels for the vertices
Text text1 = new Text();
for(int k=0; k<totalVertices; k++){
            String node =String.valueOf(SetVertices.get(k)).substring(0,2);
            text1 = new Text();
            text1.setText(node);
            text1.xProperty().bind(circ[k].centerXProperty().subtract(6));
            text1.yProperty().bind(circ[k].centerYProperty().add(6));
            text1.setFont(Font.font("Courier",FontWeight.BOLD,20));
            pane.getChildren().add(text1); // Add text1 to the pane
}
```

```
        Text text2 = new Text();
        for(int k=0; k<totalVertices2; k++){
                String node_2 =String.valueOf(SetVertices2.get(k)).substring(0,2);
                text2 = new Text();
                text2.setText(node_2);
                text2.xProperty().bind(circ2[k].centerXProperty().subtract(6));
                text2.yProperty().bind(circ2[k].centerYProperty().add(6));
                text2.setFont(Font.font("Courier",FontWeight.BOLD,20));
                pane2.getChildren().add(text2); // Add text1 to the pane
        }

        vertBox.getChildren().add(pane);
        vertBox.getChildren().add(pane2);
        // Create a scene and place it in the stage
        Scene scene = new Scene(vertBox, 720, 660);
        if(check==1){
                primaryStage.setTitle("The two graphs are not similar"); // Set the stage
title
        }
        else
        {
                primaryStage.setTitle("The two graphs are similar"); // Set the stage title
        }
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
}
public static int Find (int vertex){
        if(PARENT.get(vertex)==vertex)
                return PARENT.get(vertex);

        else
                return Find(PARENT.get(vertex));
}
public static void Union(int root1, int root2) {
        int num;
        num=RANK.get(root2);
        if( RANK.get(root1) > RANK.get(root2))
                PARENT.put(root1, root2);
        else if(RANK.get(root1) < RANK.get(root2))
                PARENT.put(root2, root1);
        else
                PARENT.put(root2, root1);RANK.put(root2,num++);
}

public static Edge[] minTree(Edge[] ed,int v, int e) {
```

```
            //Sorting edges
            for(int i=0;i<e;i++)
            for(int j=0;j<e-1;j++)
            {
                  if(ed[j].wt>ed[j+1].wt)
                   {
                        Edge t=new Edge();
                     t=ed[j];
                     ed[j]=ed[j+1];
                     ed[j+1]=t;
                  }
            }

            for(int i=0;i<v;i++) {
                  int x=Integer.parseInt(String.valueOf(SetVertices.get(i)).substring(0,1));
                  PARENT.put(x,x);
                  RANK.put(x,0);
            }
         for (int i=0;i<e;i++) {
                  int
root1=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v1.a)).substring(0,1)));
                  int
root2=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v2.a)).substring(0,1)));
                  if(root1!=root2){
                        ed[i].con=1;
                        Union(root1,root2);
                  }
         }
                  return ed;
                  }

public static void main(String[] args) throws IOException{
                     FileReader file = new FileReader("inputfile");
                      BufferedReader in = new BufferedReader(file);
                      v=Integer.parseInt(in.readLine());
                      e=Integer.parseInt(in.readLine());
                      float w1=0,w2=0;
                      int common_chk1=0,common_chk2=2;
                      for(int j=0;j<v;j++){
                            Vertex x = new Vertex(in.readLine());
                            //add it one by one to the set of vertices
                            SetVertices.add(x);
                            }
                      for(int i=0;i<e;i++){
                            ed[i]=new Edge();
                            ed[i].v1.a=Integer.parseInt(in.readLine());
```

```
                    ed[i].v2.a=Integer.parseInt(in.readLine());
                    ed[i].wt=Float.parseFloat(in.readLine());
                    ed[i].con=0;
                    }
                ed=minTree(ed,v,e);
                int m=0;
                for(int i=0;i<e;i++){
                    if(ed[i].con==1){
                        System.out.println(ed[i].v1.a + " "+ed[i].v2.a+" "+ed[i].wt);
                        min_ed[m]=ed[i];
                        w1+=ed[i].wt;
                        m++;
                        }
                    }
                //starting to draw the graph

                Scanner sc = new Scanner(in);
                List<java.awt.Point> graf = new ArrayList<java.awt.Point>();
                while (sc.hasNextInt()) {
                    graf.add(new java.awt.Point(sc.nextInt(), sc.nextInt()));
                    }
                in.close();
                sc.close();
                gr=graf;

                file = new FileReader("input");
                in = new BufferedReader(file);
                v2=Integer.parseInt(in.readLine());
                e2=Integer.parseInt(in.readLine());
                for(int j=0;j<v2;j++){
                    Vertex x = new Vertex(in.readLine());
                    //add it one by one to the set of vertices
                    SetVertices2.add(x);
            }
        for(int i=0;i<e2;i++){
            ed2[i]=new Edge();
            ed2[i].v1.a=Integer.parseInt(in.readLine());
            ed2[i].v2.a=Integer.parseInt(in.readLine());
            ed2[i].wt=Float.parseFloat(in.readLine());
            ed2[i].con=0;
            }
        ed2=minTree(ed2,v2,e2);
        m=0;
        for(int i=0;i<e2;i++){
            if(ed2[i].con==1){
```

```
                    System.out.println(ed2[i].v1.a + " "+ed2[i].v2.a+" "+ed2[i].wt);
                    min_ed2[m]=ed2[i];
                    w2+=ed2[i].wt;
                    m++;
                    }
            }
        //starting to draw the graph

        sc = new Scanner(in);
        List<java.awt.Point> graf2 = new ArrayList<java.awt.Point>();
        while (sc.hasNextInt()) {
            graf2.add(new java.awt.Point(sc.nextInt(), sc.nextInt()));
            }
        in.close();
        sc.close();
        gr2=graf2;

        check=0;
        if(v!=v2){
            check=1;
        }
        else if(w1!=w2) {
            check=1;
        }
        else {
            for (int i=0;i<v-1;i++)
            {
                if(min_ed[i].wt!=min_ed2[i].wt)
                {
                    check=1;
                    break;
                }
            }
        }

        if (check==0)
        {
            for (int i=0;i<v-1;i++)
            {
                for(int j=i+1;j<v-1;j++)
                {
                    if(min_ed[i].v1.a==min_ed[j].v1.a ||
min_ed[i].v1.a==min_ed[j].v2.a ||
                        min_ed[i].v2.a==min_ed[j].v1.a ||
min_ed[i].v2.a==min_ed[j].v2.a)
                        common_chk1=1;
```

```
                    if(min_ed2[i].v1.a==min_ed2[j].v1.a                    ||
min_ed2[i].v1.a==min_ed2[j].v2.a ||
                                        min_ed2[i].v2.a==min_ed2[j].v1.a    ||
min_ed2[i].v2.a==min_ed2[j].v2.a)
                            common_chk2=1;
                    else
                            common_chk2=0;
                    if(common_chk1!=common_chk2){
                            check=1;
                            break;
                    }
                    common_chk1=0;common_chk2=2;
                }
            }

        }
            Application.launch(args);
              }
            }
```

APPENDIX C

CODE SNIPPET FOR COMPARING TWO TREES WITH LEADER NODE

This uses the same vertex class created above.

```
package application;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import  java.util.Map;
import java.util.HashMap;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

class Edge
{
        class vertex
        {
                int a;
        }
```

```
        vertex v1;
        vertex v2;
        float wt;
        int con;
        Edge()
        {
                v1=new vertex();
                v2=new vertex();
        }
}


public class Main extends Application {
        static int v,e,v2,e2,check=0;
        static int ar[][]= new int [3][100];
        static int ar2[][]= new int [3][100];
        static Edge min_ed[]=new Edge[200];
        static Edge min_ed2[]=new Edge[200];
        static List<java.awt.Point> gr;
        static List<java.awt.Point> gr2;
        static Edge ed[]=new Edge[200];
        static Edge ed2[]=new Edge[200];
        static Map<Integer, Integer> PARENT= new HashMap();
        static Map<Integer, Integer> RANK= new HashMap();
        static int ind1=-1,ind2=-1;
        //create the set of vertices
        static List<Vertex> SetVertices = new ArrayList<Vertex>();
        static List<Vertex> SetVertices2 = new ArrayList<Vertex>();
        private Pane pane;
        private Pane pane2;
        private VBox vertBox;
        @Override
        public void start(Stage primaryStage) {
                // Create a pane to hold the graph
                pane = new Pane();
                pane2=new Pane();
                vertBox = new VBox(10);
                // retrieve the number of nodes
                int totalVertices = gr.size();
                int totalVertices2 = gr2.size();
                // define the array of nodes as colored circles
                Circle circ[] = new Circle[100];
                Circle circ2[] = new Circle[100];
                // define and draw each node in the graph
                java.awt.Point node1;
                for(int k=0; k<totalVertices; k++)
```

```
        {
                node1 = gr.get(k);
                // define a generic circle for each node
                circ[k] = new Circle();
                circ[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
                circ[k].setStroke(Color.BLACK);
                circ[k].setFill(Color.BLACK);
                circ[k].setCenterX(node1.x);
                circ[k].setCenterY(node1.y);
                // Add the circle to the pane
                pane.getChildren().add(circ[k]);
        }

        java.awt.Point node2;
        for(int k=0; k<totalVertices2; k++)
        {
                node2 = gr2.get(k);
                // define a generic circle for each node
                circ2[k] = new Circle();

circ2[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
                circ2[k].setStroke(Color.BLACK);
                circ2[k].setFill(Color.BLACK);
                circ2[k].setCenterX(node2.x);
                circ2[k].setCenterY(node2.y);
                // Add the circle to the pane
                pane2.getChildren().add(circ2[k]);
        }

        //draw the graph

        Line line2 = new Line();
        for(int k=0; k<e; k++)
        {
                // create the arc between node k and its adjacent nodes
                float w=ed[k].wt;
                int n=ed[k].v1.a;
                int l=ed[k].v2.a;
                line2 = new Line();
                line2.startXProperty().bind(circ[n].centerXProperty());
                line2.startYProperty().bind(circ[n].centerYProperty());
                line2.endXProperty().bind(circ[l].centerXProperty());
                line2.endYProperty().bind(circ[l].centerYProperty());
                line2.setStrokeWidth(w);
                line2.setStroke(Color.BLACK);
                // Add the arc to the pane
```

```
            pane.getChildren().add(line2);
}

Line line22 = new Line();
for(int k=0; k<e2; k++)
{
            // create the arc between node k and its adjacent nodes
            float w=ed2[k].wt;
            int n=ed2[k].v1.a;
            int l=ed2[k].v2.a;
            line22 = new Line();
            line22.startXProperty().bind(circ2[n].centerXProperty());
            line22.startYProperty().bind(circ2[n].centerYProperty());
            line22.endXProperty().bind(circ2[l].centerXProperty());
            line22.endYProperty().bind(circ2[l].centerYProperty());
            line22.setStrokeWidth(w);
            line22.setStroke(Color.BLACK);
            // Add the arc to the pane
            pane2.getChildren().add(line22);
}


// draw the edges in the graph
Line line1 = new Line();
for(int k=0; k<totalVertices-1; k++)
{
            // create the arc between node k and its adjacent nodes
            float w=min_ed[k].wt;
            int n=min_ed[k].v1.a;
            int l=min_ed[k].v2.a;
            line1 = new Line();
            line1.startXProperty().bind(circ[n].centerXProperty());
            line1.startYProperty().bind(circ[n].centerYProperty());
            line1.endXProperty().bind(circ[l].centerXProperty());
            line1.endYProperty().bind(circ[l].centerYProperty());
            line1.setStrokeWidth(w);
            line1.setStroke(Color.BLUE);
            // Add the arc to the pane
            pane.getChildren().add(line1);

}

Line line12 = new Line();
for(int k=0; k<totalVertices2-1; k++)
{
            // create the arc between node k and its adjacent nodes
```

```
                float w=min_ed2[k].wt;
                int n=min_ed2[k].v1.a;
                int l=min_ed2[k].v2.a;
                line12 = new Line();
                line12.startXProperty().bind(circ2[n].centerXProperty());
                line12.startYProperty().bind(circ2[n].centerYProperty());
                line12.endXProperty().bind(circ2[l].centerXProperty());
                line12.endYProperty().bind(circ2[l].centerYProperty());
                line12.setStrokeWidth(w);
                line12.setStroke(Color.BLUE);
                // Add the arc to the pane
                pane2.getChildren().add(line12);
        }

        // re-draw the nodes in the graph to overlap over the edges
        for(int k=0; k<totalVertices; k++){
                node1 = gr.get(k);
                // re-define a generic circle for each node
                circ[k] = new Circle();
                circ[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
                circ[k].setStroke(Color.BLACK);
                circ[k].setFill(Color.WHITE);
                circ[k].setCenterX(node1.x);
                circ[k].setCenterY(node1.y);
                // Add the circle to the pane
                pane.getChildren().add(circ[k]);
                }

        for(int k=0; k<totalVertices2; k++){
                node2 = gr2.get(k);
                // re-define a generic circle for each node
                circ2[k] = new Circle();

circ2[k].radiusProperty().bind(vertBox.heightProperty().divide(45));
                circ2[k].setStroke(Color.BLACK);
                circ2[k].setFill(Color.WHITE);
                circ2[k].setCenterX(node2.x);
                circ2[k].setCenterY(node2.y);
                // Add the circle to the pane
                pane2.getChildren().add(circ2[k]);
                }

        // draw the leader node with red
        node1 = gr.get(ind1);
        circ[ind1] = new Circle();
        circ[ind1].radiusProperty().bind(vertBox.heightProperty().divide(45));
```

```
circ[ind1].setStroke(Color.RED);
circ[ind1].setStrokeWidth(2);
circ[ind1].setFill(Color.WHITE);
circ[ind1].setCenterX(node1.x);
circ[ind1].setCenterY(node1.y);
// Add the circle to the pane
pane.getChildren().add(circ[ind1]);

// draw the leader node with red
                node2 = gr2.get(ind2);
                circ2[ind2] = new Circle();

circ2[ind2].radiusProperty().bind(vertBox.heightProperty().divide(45));
                circ2[ind2].setStroke(Color.RED);
                circ2[ind2].setStrokeWidth(2);
                circ2[ind2].setFill(Color.WHITE);
                circ2[ind2].setCenterX(node2.x);
                circ2[ind2].setCenterY(node2.y);
                // Add the circle to the pane
                pane2.getChildren().add(circ2[ind2]);
        // Read and display the labels for the vertices
        Text text1 = new Text();
        for(int k=0; k<totalVertices; k++){
                String node =String.valueOf(SetVertices.get(k)).substring(0,2);
                text1 = new Text();
                text1.setText(node);
                text1.xProperty().bind(circ[k].centerXProperty().subtract(6));
                text1.yProperty().bind(circ[k].centerYProperty().add(6));
                text1.setFont(Font.font("Courier",FontWeight.BOLD,20));
                pane.getChildren().add(text1); // Add text1 to the pane
        }

        Text text2 = new Text();
        for(int k=0; k<totalVertices2; k++){
                String node_2 =String.valueOf(SetVertices2.get(k)).substring(0,2);
                text2 = new Text();
                text2.setText(node_2);
                text2.xProperty().bind(circ2[k].centerXProperty().subtract(6));
                text2.yProperty().bind(circ2[k].centerYProperty().add(6));
                text2.setFont(Font.font("Courier",FontWeight.BOLD,20));
                pane2.getChildren().add(text2); // Add text1 to the pane
        }

        vertBox.getChildren().add(pane);
        vertBox.getChildren().add(pane2);
        // Create a scene and place it in the stage
```

```
                Scene scene = new Scene(vertBox, 720, 660);
                if(check==1){
                        primaryStage.setTitle("The two graphs are not similar"); // Set the
stage title
                }
                else
                {
                        primaryStage.setTitle("The two graphs are similar"); // Set the
stage title
                }
                primaryStage.setScene(scene); // Place the scene in the stage
                primaryStage.show(); // Display the stage
        }

        public static boolean check_equal(int a,int b) {
                System.out.println("start  "+a+ "  "+b);
                int index1=-1,index2=-1,other1=-1,other2=-1,index=0;
                boolean result=true;
                float chk1,chk2;
                for(int i=0;i<v;i++)
                {
                        if(ar[0][i]==a && ar[2][i]==0){
                                index1=i;
                                ar[2][i]=1;
                                break;
                        }

                }
                for(int j=0;j<v;j++)
                {
                        if(ar2[0][j]==b && ar2[2][j]==0){
                                index2=j;
                                ar2[2][j]=1;
                                break;
                        }
                }
                if(index1==-1) {
                        return true;
                }

        System.out.println(String.valueOf(SetVertices.get(index1)).substring(0,1));
                if(ar[1][index1]==0 && ar2[1][index2]==0)
                        return true;
                else {
                        for(int i=0;i<v-1;i++)
                        {
```

```
                    chk1=-1;chk2=-1;

        if(Integer.parseInt(String.valueOf(SetVertices.get(min_ed[i].v1.a)).substring(0,1))
==a){
                        chk1=min_ed[i].wt;
                        other1=min_ed[i].v2.a;
                    }
                    else                                                    if
(Integer.parseInt(String.valueOf(SetVertices.get(min_ed[i].v2.a)).substring(0,1))==a){
                        chk1=min_ed[i].wt;
                        other1=min_ed[i].v1.a;
                    }

        if(Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[i].v1.a)).substring(0,1
))==b){
                        chk2=min_ed2[i].wt;
                        other2=min_ed2[i].v2.a;
                    }
                    else                                                    if
(Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[i].v2.a)).substring(0,1))==b){
                        chk2=min_ed2[i].wt;
                        other2=min_ed2[i].v1.a;
                    }
                    if(chk1!=chk2){
                        index=1;
                        break;
                    }
                    else if(chk1!=-1) {
                        if(ar[2][other1]==0 &&
                                ar2[2][other2]==0)
                                result=result                               &&
check_equal(Integer.parseInt(String.valueOf(SetVertices.get(other1)).substring(0,1)),

        Integer.parseInt(String.valueOf(SetVertices.get(other2)).substring(0,1)));
                    }
                }
                if (index==1)
                    return false;
                else
                    return result;
            }
        }

        public static int Find (int vertex){
        if(PARENT.get(vertex)==vertex)
            return PARENT.get(vertex);
```

```
        else
                return Find(PARENT.get(vertex));
}
public static void Union(int root1, int root2) {
        int num;
        num=RANK.get(root2);
        if( RANK.get(root1) > RANK.get(root2))
                PARENT.put(root1, root2);
        else if(RANK.get(root1) < RANK.get(root2))
                PARENT.put(root2, root1);
        else
                PARENT.put(root2, root1);RANK.put(root2,num++);
}

public static Edge[] minTree(Edge[] ed,int v, int e) {
           //Sorting edges
           for(int i=0;i<e;i++)
           for(int j=0;j<e-1;j++)
           {
                if(ed[j].wt>ed[j+1].wt)
                 {
                        Edge t=new Edge();
                   t=ed[j];
                   ed[j]=ed[j+1];
                   ed[j+1]=t;
                }
           }

           for(int i=0;i<v;i++) {
                int x=Integer.parseInt(String.valueOf(SetVertices.get(i)).substring(0,1));
                PARENT.put(x,x);
                RANK.put(x,0);
           }
        for (int i=0;i<e;i++) {
                int
root1=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v1.a)).substring(0,1)));
                int
root2=Find(Integer.parseInt(String.valueOf(SetVertices.get(ed[i].v2.a)).substring(0,1)));
                if(root1!=root2){
                        ed[i].con=1;
                        Union(root1,root2);
                }
        }
                return ed;
                }
```

```
        public static void main(String[] args) throws IOException {
                int lead1=-1,lead2=-1;
                int degree=0;
                FileReader file = new FileReader("inputfile");
                BufferedReader in = new BufferedReader(file);
                v=Integer.parseInt(in.readLine());
                e=Integer.parseInt(in.readLine());
                lead1=Integer.parseInt(in.readLine());
                float w1=0,w2=0;
                for(int j=0;j<v;j++){
                        Vertex x = new Vertex(in.readLine());
        //              add it one by one to the set of vertices
                        SetVertices.add(x);
                        ar[0][j]=Integer.parseInt(x.getName());
                        if(ar[0][j]==lead1)
                                ind1=j;
                }
                for(int i=0;i<e;i++){
                        ed[i]=new Edge();
                        ed[i].v1.a=Integer.parseInt(in.readLine());
                        ed[i].v2.a=Integer.parseInt(in.readLine());
                        ed[i].wt=Float.parseFloat(in.readLine());
                        ed[i].con=0;
                }
                ed=minTree(ed,v,e);
                int m=0;
                for(int i=0;i<e;i++){
                        if(ed[i].con==1){
                                System.out.println(ed[i].v1.a      +      "      "+ed[i].v2.a+"
"+ed[i].wt);
                                min_ed[m]=ed[i];
                                w1+=ed[i].wt;
                                m++;
                        }
                }

                for(int i=0;i<v;i++){
                        degree=0;
                        for(int j=0;j<v-1;j++) {

        if(ar[0][i]==Integer.parseInt(String.valueOf(SetVertices.get(min_ed[j].v1.a)).subs
tring(0,1))
                                                        ||
ar[0][i]==Integer.parseInt(String.valueOf(SetVertices.get(min_ed[j].v2.a)).substring(0,1)
))
```

```
                                        degree++;
                                }
                                ar[1][i]=degree;
                                ar[2][i]=0;
                        }
        //      starting to draw the graph
                        Scanner sc = new Scanner(in);
                        List<java.awt.Point> graf = new ArrayList<java.awt.Point>();
                        while (sc.hasNextInt()) {
                                graf.add(new java.awt.Point(sc.nextInt(), sc.nextInt()));
                        }
                        in.close();
                        sc.close();
                        gr=graf;
                        file = new FileReader("input");
                        in = new BufferedReader(file);
                        v2=Integer.parseInt(in.readLine());
                        e2=Integer.parseInt(in.readLine());
                        lead2=Integer.parseInt(in.readLine());
                        for(int j=0;j<v2;j++){
                                Vertex x = new Vertex(in.readLine());
                //      add it one by one to the set of vertices
                                SetVertices2.add(x);
                                ar2[0][j]=Integer.parseInt(x.getName());
                                if(ar2[0][j]==lead2)
                                        ind2=j;
                        }
                        for(int i=0;i<e2;i++){
                                ed2[i]=new Edge();
                                ed2[i].v1.a=Integer.parseInt(in.readLine());
                                ed2[i].v2.a=Integer.parseInt(in.readLine());
                                ed2[i].wt=Float.parseFloat(in.readLine());
                                ed2[i].con=0;
                        }
                        ed2=minTree(ed2,v2,e2);
                        m=0;
                        for(int i=0;i<e2;i++){
                                if(ed2[i].con==1){
                                        System.out.println(ed2[i].v1.a      +      "      "+ed2[i].v2.a+"
"+ed2[i].wt);
                                        min_ed2[m]=ed2[i];
                                        //min_ed2[m].v1.a=
                                        w2+=ed2[i].wt;
                                        m++;
                                }
                        }
```

```
                for(int i=0;i<v2;i++){
                        degree=0;
                        for(int j=0;j<v2-1;j++) {

        if(ar2[0][i]==Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[j].v1.a)).su
bstring(0,1))
                                                        ||
ar2[0][i]==Integer.parseInt(String.valueOf(SetVertices.get(min_ed2[j].v2.a)).substring(0,
1))
                                                        )
                                                degree++;
                        }
                        ar2[1][i]=degree;
                        ar2[2][i]=0;
                }
//      starting to draw the graph

                sc = new Scanner(in);
                List<java.awt.Point> graf2 = new ArrayList<java.awt.Point>();
                while (sc.hasNextInt()) {
                        graf2.add(new java.awt.Point(sc.nextInt(), sc.nextInt()));
                }
                in.close();
                sc.close();
                gr2=graf2;

                check=0;
                if(v!=v2){
                        check=1;
                }
                else if(w1!=w2) {
                        check=1;
                }
                else {
                        for (int i=0;i<v-1;i++) {
                                if(min_ed[i].wt!=min_ed2[i].wt) {
                                        check=1;
                                        break;
                                }
                        }
                }
                if (check==0) {
                        if (!check_equal(lead1,lead2))
                                check=1;
                }
```

```
        System.out.println(check);

        Application.launch(args);
        }
}
```

BIBLIOGRAPHY

[1] Ghoniem, M.; Fekete, J.; Castagliola, P., "A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations," *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on* , vol., no., pp.17,24.

[2] (Horst, et al., 2002, pp. 123-132)

[3] Wang, J.T.L.; Shapiro, B.A.; Shasha, D.; Kaizhong Zhang; Currey, K.M., "An algorithm for finding the largest approximately common substructures of two trees," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.20, no.8, pp.889,895, Aug 1998.

[4] Blasch, E.; Plano, S., "DFIG Level 5 (User Refinement) issues supporting Situational Assessment Reasoning," *Information Fusion, 2005 8th International Conference on* , vol.1, no., pp.xxxv,xliii, 25-28 July 2005.

[5] Perry, R.P.; DiPietro, R.C.; Fante, R., "SAR imaging of moving targets," *Aerospace and Electronic Systems, IEEE Transactions on* , vol.35, no.1, pp.188,200, Jan 1999.

[6] Schulz, D.; Burgard, W.; Fox, D.; Cremers, A.B., "Tracking multiple moving targets with a mobile robot using particle filters and statistical data association," *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on* , vol.2, no., pp.1665,1670 vol.2, 2001

[7] Raymond; John, W.; Gardiner; Eleanor, J.; Willet; Peter," Calculation of Graph Similarity using Maximum Common Edge Subgraphs," The Computer Journal, 2002, vol.45, no.6, pp.631,644.

[8] Papadopoulos, A.N.; Manolopoulos, Y., "Structure-based similarity search with graph histograms," *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on* , vol., no., pp.174,178, 1999.

[9] Srivastav, A.; Yicheng Wen; Hendrick, E.; Chattopadhyay, I.; Ray, A.; Phoha, S., "Information fusion for object & situation assessment in sensor networks," *American Control Conference (ACC), 2011* , vol., no., pp.1274,1279, June 29 2011-July 1 2011.

[10] Pek Hui, F.; Gee Wah, NG.,"High- level Information Fusion: An Overview," Advances in Information Fusion, 2013, vol.8, no.1, pp.33,72, 2013.

[11] John Salerno; Mike Hinman; Dough Boulware; Paul Bello,"Information Fusion for Situational Awareness," ISIF, 2003, vol., no., pp.507,513, 2003.

[12] Erick Blasch; Ivan Kadar; John Salerno; Mieczyslaw Kokar, M.; Subrata Das; Gerald Powell , M.; Daniel Corkill D.; Enrique Ruspini H.," Issues and Challenges in Situation Assessment (Level 2 Fusion)," Journal of Advances in Information Fusion (JAIF), 2006, vol.1, no.2, pp.122,139, 2006.

[13] Joint Operation Planning (JP 5-0), Washington, DC: US Government Publishing Office, 2006.

[14] E. Blasch, I. Kadar, K. Hintz, J. Bierman, C. Chong and S. Das, "Resource Management Coordination with Level 2/3 Fusion Issues and Challenges," IEEE Aerospace and Electronic Systems Magazine, vol. 23, no. 3, pp. 32-46, 2008.

[15] F. Lanchester, "Aircraft in warfare: the dawn of the fourth arm," Engineering, pp. 422-423, 1914.

[16] R. Brown, "Theory of Combat: The Probability of Winning," Operations Research, vol. 11, no. 3, pp. 418-425, 1993.

[17] D. Hall and J. Llinas, Multisensor Data Fusion Handbook, CRC Press, 2007.

[18] P. Hester and A. Tolk, "Using Lanchester's equations for sequential battle prediction enabling better military decision support," Int. Jorunal of Defense Support Systems, vol. 2, pp. 76-90, 2009.

[19] K. Trivedi, Probability and Statistics with Reliability, Queing and Computer Science Applications, New York: Wiley-Interscience, 2002.

[20] A. Washburn and M. Kress, Combat Modeling, Springer, 2009.

[21] X. Zhua, Y. Yuanb, C. Rorresc and M. Kamb, "Distributed M-ary hypothesis testing with binary local decisions," *Information Fusion,* vol. 5, no. 3, pp. 157-167, 2004.

[22] M. L. Fernandez and G. Valiente, "A graph distance metric combining maximum common subgraph and minimum common supergraph," *Pattern Recognition Letters,* vol. 22, pp. 753-758, 2001.

[23] Bunke, H.," On a relation between graph edit distance and maximum common subgraph," Elsevier science, 1997.

[24] http://en.wikipedia.org/wiki/Kruskal%27s_algorithm

VITA

Graduate College

University of Nevada, Las Vegas

Pavan Kumar Pallapunidi

Degrees:

Bachelor of Technology in Information Technology, 2010

Jawaharlal Nehru Technological University, Hyderabad, India.

Master of Science in Computer Science, 2015

University of Nevada Las Vegas

Thesis Title: Situational Assessment using Graph Comparison

Thesis Examination Committee:

Chair Person, Dr. Wolfgang Bein, Ph.D.

Committee Member, Dr. Ajoy K. Datta, Ph.D

Committee Member, Dr. Ju-Yeon Jo, Ph.D.

Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.