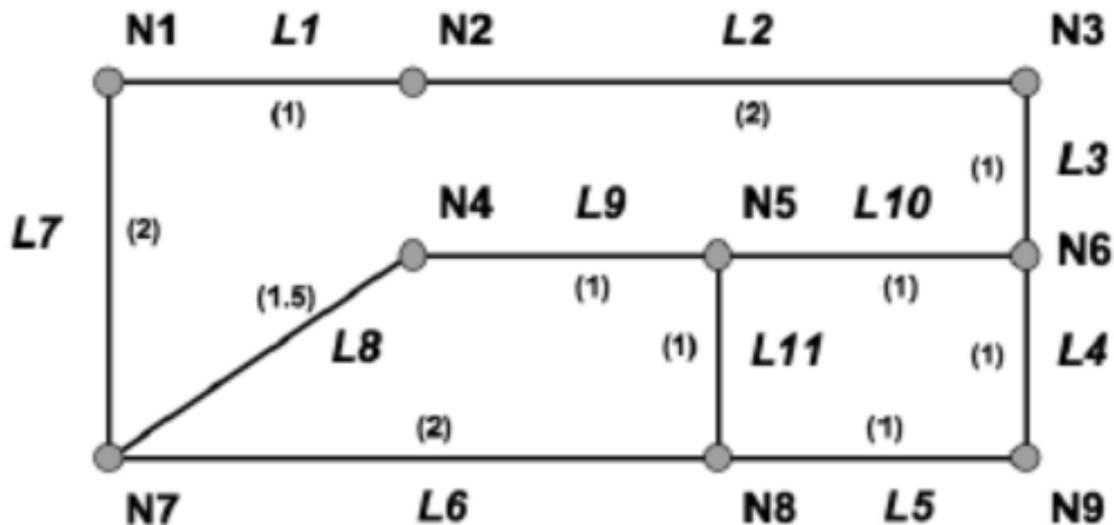


## REQUIREMENTS

First we will create UNET network. This is a simple network with undirected links. Links have a cost proportional to their length; the cost of each link is shown in parentheses. We will use the network later to illustrate network analysis functions.



The CREATE\_SDO\_NETWORK operator given below creates all structures of a network and populates required information into the USER\_SDO\_NETWORK\_METADATA.

**BEGIN**

```
SDO_NET.CREATE_SDO_NETWORK (
  NETWORK                => 'UNET',
  NO_OF_HIERARCHY_LEVELS => 1,
  IS_DIRECTED           => FALSE,
  NODE_TABLE_NAME       => 'UNET_NODES',
  NODE_GEOM_COLUMN      => 'GEOM',
  NODE_COST_COLUMN      => NULL,
  LINK_TABLE_NAME       => 'UNET_LINKS',
  LINK_COST_COLUMN      => 'COST',
  LINK_GEOM_COLUMN      => 'GEOM',
  PATH_TABLE_NAME       => 'UNET_PATHS',
  PATH_GEOM_COLUMN      => 'GEOM',
  PATH_LINK_TABLE_NAME  => 'UNET_PLINKS'
);
END;
```

SQL statements given below populates data into UNET\_NODES and UNET\_LINKS tables.

**-- Populate the node table**

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (1, 'N1', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (1,3,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (2, 'N2', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (2,3,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (3, 'N3', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,3,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (4, 'N4', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (2,2,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (5, 'N5', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (3,2,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (6, 'N6', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,2,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (7, 'N7', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (1,1,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (8, 'N8', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (3,1,NULL), null, null));
```

```
INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (9, 'N9', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,1,NULL), null, null));
```

**-- Populate the link table**

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
VALUES ( 1, 'L1', 1, 2, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1), SDO_ORDINATE_ARRAY (1,3, 2,3)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
VALUES ( 2, 'L2', 2, 3, 2, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1), SDO_ORDINATE_ARRAY (2,3, 4,3)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
VALUES ( 3, 'L3', 3, 6, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1), SDO_ORDINATE_ARRAY (4,3, 4,2)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
VALUES ( 4, 'L4', 6, 9, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1), SDO_ORDINATE_ARRAY (4,2, 4,1)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES ( 5, 'L5', 9, 8, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (4,1, 3,1)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES ( 6, 'L6', 8, 7, 2, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (3,1, 1,1)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES ( 7, 'L7', 7, 1, 2, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (1,1, 1,3)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES ( 8, 'L8', 7, 4, 1.5, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (1,1, 2,2)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES ( 9, 'L9', 4, 5, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (2,2, 3,2)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES (10, 'L10', 5, 6, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (3,2, 4,2)));
```

```
INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
```

```
VALUES (11, 'L11', 5, 8, 1, SDO_GEOMETRY (2002, NULL, NULL, SDO_ELEM_INFO_ARRAY (1,2,1),  
SDO_ORDINATE_ARRAY (3,2, 3,1)));
```

## CREATION OF THE MAIN FORM

- Create a new Project named NetworkAnalysis.
- Add a new JFrameForm named NetworkMain to the Project.
- Add required jar files into the Project to be able to work with Oracle.
- Design your main form as shown in Figure-1.

## DATABASE PANEL

Add a JTabbedPane into your main form window and name it "Database". Name your form elements as follows: Main form is separated into two parts. The top part holds various panels on reached by clicking tabs and the bottom part holds a panel including text area for general output messages. Top panel name is JPanelDB and inside this panel put a button named JBtnConnect. The panel at the bottom named JPanelText holds a text area object named JTextOutput.

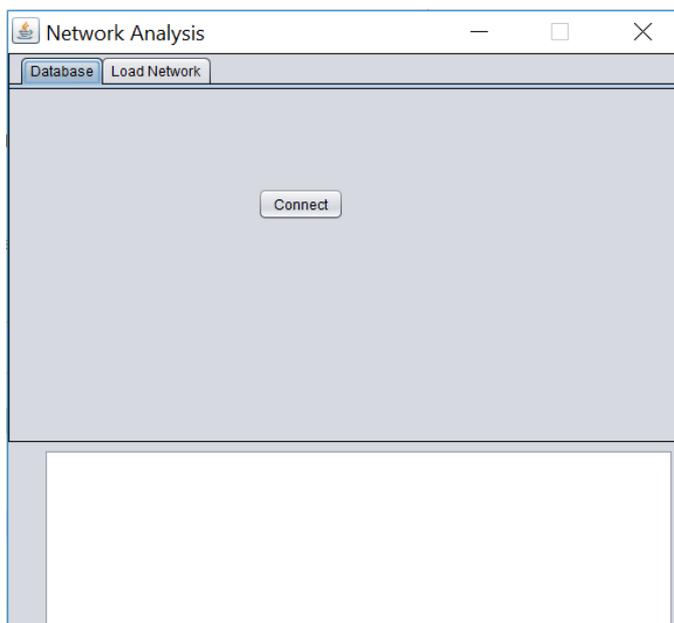


Figure-1 "Database" panel form design

Define Connection object "con" in the class NetworkMain.

```
package networkanalysis;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author SPLASH
 */
public class NetworkMain extends javax.swing.JFrame {
    private static Connection con;

    /**
     * Creates new form NewJFrame
     */
    public NetworkMain() {
        initComponents();
    }
}
```

Code Part-1 Definition of Connection object

Add action click method of "Connect" button and put required code to connect to database as shown in Code Part-2. In this method you call openConnection and closeConnection methods define as shown in Code Part-3.

```
private void btnConnectActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if(con == null)
        {
            openConnection();
            btnConnect.setText("Disconnect");
            jTextOutput.setText("Connected to database successfully...");
        }
        else
        {
            closeConnection();
            btnConnect.setText("Connect");
            jTextOutput.setText("Closed database connection...");
        }
    } catch (SQLException ex) {
        Logger.getLogger(NetworkMain.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Code Part-2 Connect button action method

```
private void openConnection() throws SQLException
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    String url = "";

    url = "jdbc:oracle:thin:@//localhost:1521/orcl";

    if(con == null)
    {
        con = DriverManager.getConnection(url, "SPATIAL", "spatial123");
        con.setAutoCommit(true);
    }
}

private void closeConnection() throws SQLException
{
    if(con != null)
    {
        con.close();
        con = null;
    }
}
```

Code Part-3 openConnection and closeConnection methods.

## LOAD NETWORK PANEL

To create this panel add a new JPanel into the JTabbedPane and name the tab as “Load Network”. In this panel add objects JPanelNetworkLoad, JTextNetworkName, jBtnLoadNetwork respectively as shown in Figure-2.

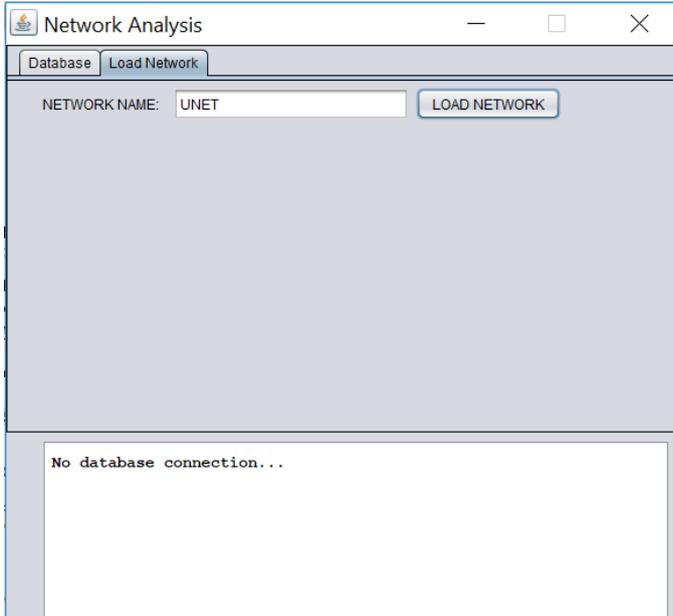


Figure-2 “Load Network” panel form design.

Define Network object “unet” in the class NetworkMain as shown in Code Part-4.

```
public class NetworkMain extends javax.swing.JFrame {  
    private static Connection con;  
    private Network unet;
```

Code Part-4 Definition of Connection object

Add action click method of “Load Network” button and put required code to load network as shown in Code Part-5.

```
private void jBtnLoadNetworkActionPerformed(java.awt.event.ActionEvent evt) {  
  
    try {  
        if(con!=null && !con.isClosed())  
        {  
            unet = NetworkManager.readNetwork(con, jtxtNetworkName.getText(), false);  
            jTextOutput.setText("Network loaded...\n");  
        }  
        else  
        {  
            jTextOutput.setText("No database connection...\n");  
        }  
    } catch (SQLException ex) {  
        jTextOutput.setText(ex.getMessage());  
    } catch (NetworkDataException ex) {  
        jTextOutput.setText(ex.getMessage());  
    }  
}
```

Code Part-5 “Load Network” button action method

**Note:** Loading a network in “read-only” mode still allows you to perform network updates in memory. However, you will not be able to apply the changes to the database. The writeNetwork() method writes a network back to the database. It actually takes only the changes you made to the memory-resident network and applies them to the persistent copy stored in the database. The main use of the writeNetwork() method is to store the paths calculated by the analysis functions into the path and path link tables in the database. To use that method, you must first have loaded the network in “update” mode.

**NETWORK ANALYSIS**

To create this panel add a new JPanel into the JTabbedPane and name the tab as “Network Analysis”. In this panel add objects JPanelNetworkAnalyse, jComboAnalyseType, JScrollPaneAnalyse respectively as shown in Figure-3. Items you should add into the combo box are “Shortest Path, Nearest Neighbors, All Nodes Within Some Distance”.

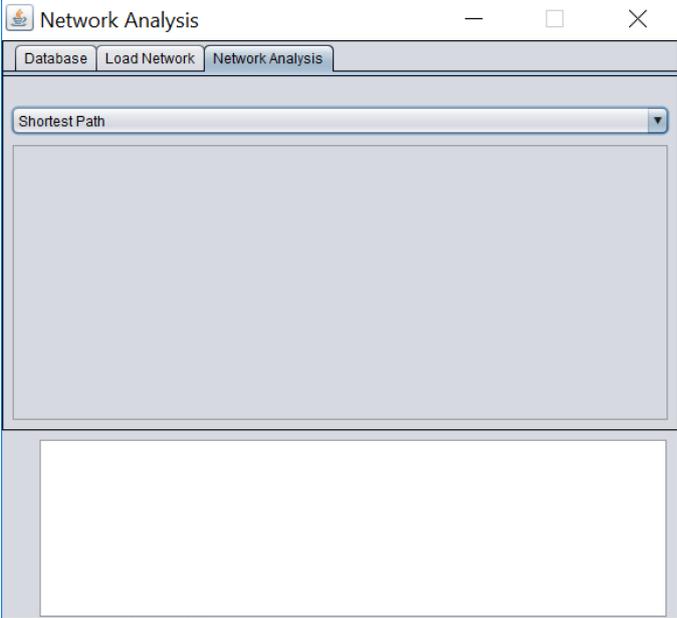


Figure-3 “Network Analysis” panel form design.

## SHORTEST PATH

The JScrollPane object added into the JPanelNetworkAnalyse will show sub forms that appears when you select an item from combo box. The sub forms are JPanelForm objects. So to add the first sub form add a new JPanelForm named formShortestPath. In this form add objects JComboBoxStartNode, JComboBoxEndNode, JButtonShortestPath respectively as shown in Figure-4.

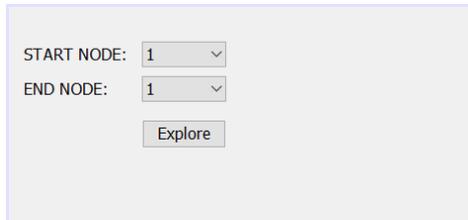


Figure-4 Form design for “Shortest Path” analysis.

Define Network, Connection and Path objects in the formShortestPath class. Design your constructor method as shown in Code Part-6.

```
public class formShortestPath extends javax.swing.JPanel {  
  
    Network net;  
    Connection cnn;  
    Path path;  
  
    /**  
     * Creates new form formShortestPath  
     * @param net  
     * @param cnn  
     */  
    public formShortestPath(Network net, Connection cnn) {  
        initComponents();  
        this.net = net;  
        this.cnn = cnn;  
    }  
}
```

Code Part-6 formShortestPath class and its constructor method.

Add a new method named findShortestPath as shown in Code Part-7.

```
private void findShortestPath(int startNode, int endNode)  
{  
    try {  
        path = NetworkManager.shortestPath(net, startNode, endNode);  
        if(path != null)  
        {  
            NetworkMain.jTextOutput.setText("Path Cost: " + path.getCost());  
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Number of Links: " + path.getNoOfLinks());  
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Simple Path? " + path.isSimple());  
  
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\nLinks Traversed:");  
            Link[] linkArray = path.getLinkArray();  
            for (int i = 0; i < linkArray.length; i++) {  
                NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Link " + linkArray[i].getID() + "\t"  
                    + linkArray[i].getName() + "\t(cost: " + linkArray[i].getCost()+")");  
            }  
  
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Nodes traversed:");  
            Node[] nodeArray = path.getNodeArray();  
            for (int i = 0; i < nodeArray.length; i++) {  
                NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Node: " + nodeArray[i].getID() + "\t"  
                    + nodeArray[i].getName() + "\t" + nodeArray[i].getCost());  
            }  
        } else {  
            NetworkMain.jTextOutput.setText("Could not find solution...");  
        }  
    } catch (NetworkDataException ex) {  
        NetworkMain.jTextOutput.setText(ex.getMessage());  
    }  
}
```

Code Part-7 “findShortestPath” method

Add action click method of “Explore” button and put required code as shown in Code Part-8. In this method you call findShortestPath method.

```
private void jButtonShortestPathActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int startNode = Integer.parseInt(jComboStartNode.getSelectedItem().toString());  
    int endNode = Integer.parseInt(jComboEndNode.getSelectedItem().toString());  
  
    findShortestPath(startNode, endNode);  
}
```

Code Part-8 “Explore” button action method

To show the sub form “formShortestPath” in the JScrollPane write the necessary code for action method of combobox in the NetworkMain class as shown in Code Part-9 and Code Part-10.

```
public class NetworkMain extends javax.swing.JFrame {  
    private static Connection con;  
    private Network unet;  
    private formShortestPath frmShortestPath;
```

Code Part-9 Adding formShortestPath object in the NetworkMain class

```
private void jComboAnalyseTypeActionPerformed(java.awt.event.ActionEvent evt) {  
    int option = jComboAnalyseType.getSelectedIndex();  
    if(option == 0)  
    {  
        frmShortestPath = new formShortestPath(unet, con);  
        jScrollPaneAnalyse.setViewportViewView(frmShortestPath);  
        frmShortestPath.setVisible(true);  
    }  
}
```

Code Part-10 “jComboAnalyseType” combobox action method

Figure 5 shows the result of shortest path analysis between node 4 and node 3.

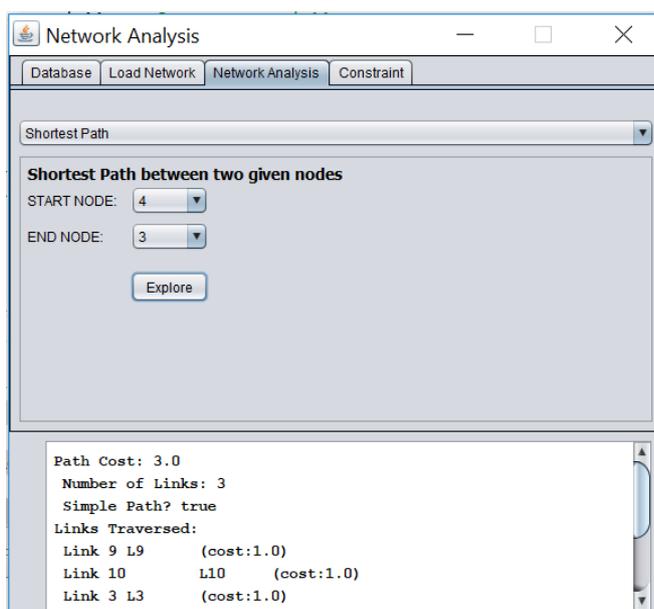


Figure-5 Shortest path analysis result

## NEAREST NEIGHBORS

The nearestNeighbors() method returns an array of Path objects. Not only does it tell you who the nearest nodes are, but it also tells you how to reach them.

Add a new JPanelForm named formNearestNeighbor. In this form add objects jComboStartNode, jtxtNumberOfNeighbor, JButtonFindNeighbors respectively as shown in Figure-6.

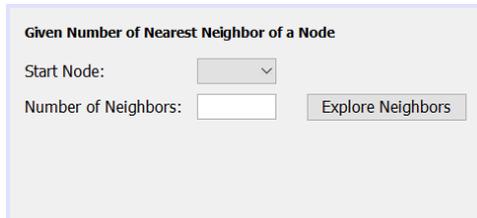


Figure-6 Form design for “Nearest Neighbor” analysis.

Define Network, Connection and Path objects in the formNearestNeighbor class. Design your constructor method as shown in Code Part-11.

```
public class formNearestNeighbors extends javax.swing.JPanel {  
  
    Network net;  
    Connection cnn;  
    Path[] paths;  
    /**  
     * Creates new form formNearestNeighbors  
     * @param net  
     * @param cnn  
     */  
    public formNearestNeighbors(Network net, Connection cnn) {  
        initComponents();  
        this.net = net;  
        this.cnn = cnn;  
    }  
}
```

Code Part-11 formNearestNeighbor class and its constructor method.

Add a new method named findNearestNeighbor as shown in Code Part-12.

```
private void findNearestNeighbor(int startNode, int numberOfNeighbor)  
{  
    try {  
        paths = NetworkManager.nearestNeighbors(net, startNode, numberOfNeighbor);  
        NetworkMain.jTextOutput.setText(paths.length+ " nearest neighbors of node "+ startNode+ " in network "+net.getName() );  
  
        if (paths != null) {  
            int i=0;  
            for (Path yol : paths) {  
                //jtxtMesaj.setText(jtxtMesaj.getText() + "\n Düşüm" + yol.getEndNode().getID() + ", Yol Maliyet:" + yol.getCost());  
                NetworkMain.jTextOutput.setText( NetworkMain.jTextOutput.getText() + "\n Node" + yol.getEndNode().getID() +  
                    ", Path Cost:" + yol.getCost());  
                i++;  
            }  
        } else {  
            NetworkMain.jTextOutput.setText("Could not find any node...");  
        }  
    } catch (NetworkDataException ex) {  
        NetworkMain.jTextOutput.setText(ex.getMessage());  
    }  
}
```

Code Part-12 “findNearestNeighbor” method.

Add action click method of “Explore Neighbors” button and put required code as shown in Code Part-13. In this method you call findNearestNeighbor method.

```

private void jButtonFindNeighborsActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int numberOfNeighbor, nodeId;
    if (!jtxtNumberOfNeighbor.getText().trim().equals("")) {
        numberOfNeighbor = Integer.parseInt(jtxtNumberOfNeighbor.getText());
        nodeId = Integer.parseInt(jComboStartNode.getSelectedItem().toString());
        findNearestNeighbor(nodeId, numberOfNeighbor);
    } else {
        NetworkMain.jTextOutput.setText("Number of neighbours does not exists");
    }
}
}

```

Code Part-13 “Explore Neighbors” button action method

To show the sub form “formNearestNeighbor” in the JScrollPane write the necessary code for action method of combobox in the NetworkMain class as shown in Code Part-14 and Code Part-15.

```

public class NetworkMain extends javax.swing.JFrame {
    private static Connection con;
    private Network unet;
    private formShortestPath frmShortestPath;
    private formNearestNeighbors frmNearestNeighbor;
}

```

Code Part-14 Adding formNearestNeighbor object in the NetworkMain class.

```

private void jComboAnalyseTypeActionPerformed(java.awt.event.ActionEvent evt) {
    int option = jComboAnalyseType.getSelectedIndex();
    if(option == 0) //Show shortest path form in JScrollPane
    {
        frmShortestPath = new formShortestPath(unet, con, constraint);
        jScrollPaneAnalyse.setViewportViewView(frmShortestPath);
        frmShortestPath.setVisible(true);
    }
    else if(option == 1)
    {
        frmNearestNeighbor = new formNearestNeighbors(unet, con);
        jScrollPaneAnalyse.setViewportViewView(frmNearestNeighbor);
        frmNearestNeighbor.setVisible(true);
    }
}

```

Code Part-15 “jComboAnalyseType” combobox action method.

Figure-7 shows the result of two nearest neighbors of node 4.

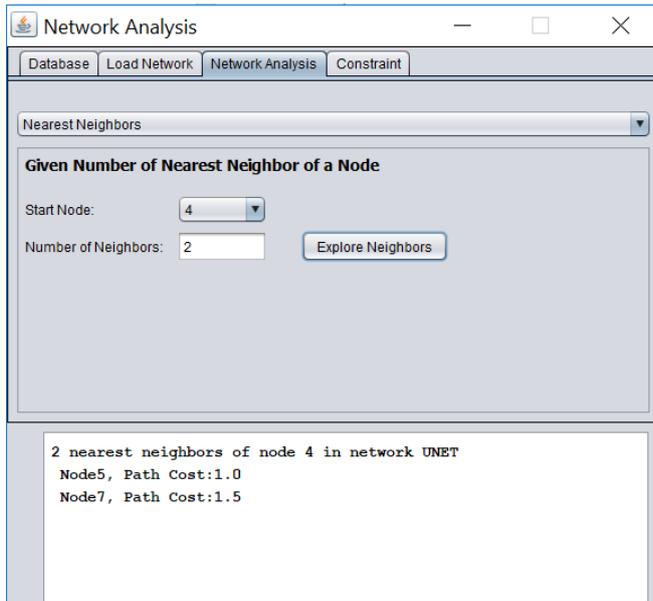


Figure-7 Nearest neighbors analysis result

## FINDING ALL NODES WITHIN SOME DISTANCE

Another common network analysis operation selects nodes based on the distance that separates them from a starting node. A typical example of using the `withinCost()` method is to find the network nodes (road intersections) that are within a certain driving distance or driving time from a given store.

Add a new `JPanelForm` named `formNodesWithinDistance`. In this form add objects `jComboStartNode`, `jtxtMaxDistance`, `jButtonFindNodesWithinDistance` respectively as shown in Figure-8.

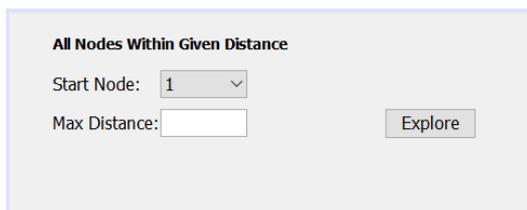


Figure-8 Form design for “All Nodes Within Some Distance” analysis

Define `Network`, `Connection`, `Path[ ]` and `Path` objects in the `formNearestNeighbor` class. Design your constructor method as shown in Code Part-16.

```

public class formNodesWithinDistance extends javax.swing.JPanel {

    Network net;
    Connection cnn;
    Path[] paths;
    Path path;
    /**
     * Creates new form formNodesWithinDistance
     * @param net
     * @param cnn
     */
    public formNodesWithinDistance(Network net, Connection cnn) {
        initComponents();
        this.net = net;
        this.cnn = cnn;
    }
}

```

Code Part-16 formNodesWithinDistance class and it's constructor method.

Add a new method named findNodesWithinDistance as shown in Code Part-17.

```

private void findNodesWithinDistance(int startNode, double distance)
{
    try {
        paths = NetworkManager.withinCost(net, startNode, distance);

        if (paths != null) {
            NetworkMain.jTextOutput.setText(paths.length + " nodes from node " + startNode + " in network " + net.getName() +
                " within a cost of " + distance);

            for(int i=0;i<paths.length;i++){
                path = paths[i];
                NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Node" + path.getEndNode().getID() +
                    ", Path Cost:" + path.getCost());
            }
        } else {
            NetworkMain.jTextOutput.setText("Could not find any node...");
        }
    } catch (NetworkDataException ex) {
        NetworkMain.jTextOutput.setText(ex.getMessage());
    }
}

```

Code Part-17 "findNodesWithinDistance" method.

Add action click method of "Explore" button and put required code as shown in Code Part-18. In this method you call findNodesWithinDistance method.

```

private void jButtonFindNodesWithinDistanceActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    double distance = 0;
    int starNode = 0;
    if (!jtxtMaxDistance.getText().trim().equals("")) {
        distance = Double.parseDouble(jtxtMaxDistance.getText().replace(',', '.'));
        starNode = Integer.parseInt(jComboStartNode.getSelectedItem().toString());
        findNodesWithinDistance(starNode, distance);
    } else {
        NetworkMain.jTextOutput.setText("Could not find any node within given distance...");
    }
}

```

Code Part-18 "Explore" button action method

To show the sub form "formNodesWithinDistance" in the JScrollPane write the necessary code for action method of combobox in the NetworkMain class as shown in Code Part-19 and Code Part-20.

```

public class NetworkMain extends javax.swing.JFrame {
    private static Connection con;
    private Network unet;
    private formShortestPath frmShortestPath;
    private formNearestNeighbors frmNearestNeighbor;
    private formNodesWithinDistance frmNodesWithinDistance;
}

```

Code Part-19 Adding formNodesWithinDistance object in the NetworkMain class.

```

private void jComboAnalyseTypeActionPerformed(java.awt.event.ActionEvent evt) {
    int option = jComboAnalyseType.getSelectedIndex();
    if(option == 0) //Show shortest path form in JscrollPane
    {
        frmShortestPath = new formShortestPath(unet, con);
        jScrollPaneAnalyse.setViewportViewView(frmShortestPath);
        frmShortestPath.setVisible(true);
    }
    else if(option == 1)
    {
        frmNearestNeighbor = new formNearestNeighbors(unet, con);
        jScrollPaneAnalyse.setViewportViewView(frmNearestNeighbor);
        frmNearestNeighbor.setVisible(true);
    }
    else if(option == 2)
    {
        frmNodesWithinDistance = new formNodesWithinDistance(unet, con);
        jScrollPaneAnalyse.setViewportViewView(frmNodesWithinDistance);
        frmNodesWithinDistance.setVisible(true);
    }
}
}

```

Code Part-20 “jComboAnalyseType” combobox action method.

Lets find the nodes that are at a distance of less than three “cost units” from node N2 on the UNET network (Figure-9)

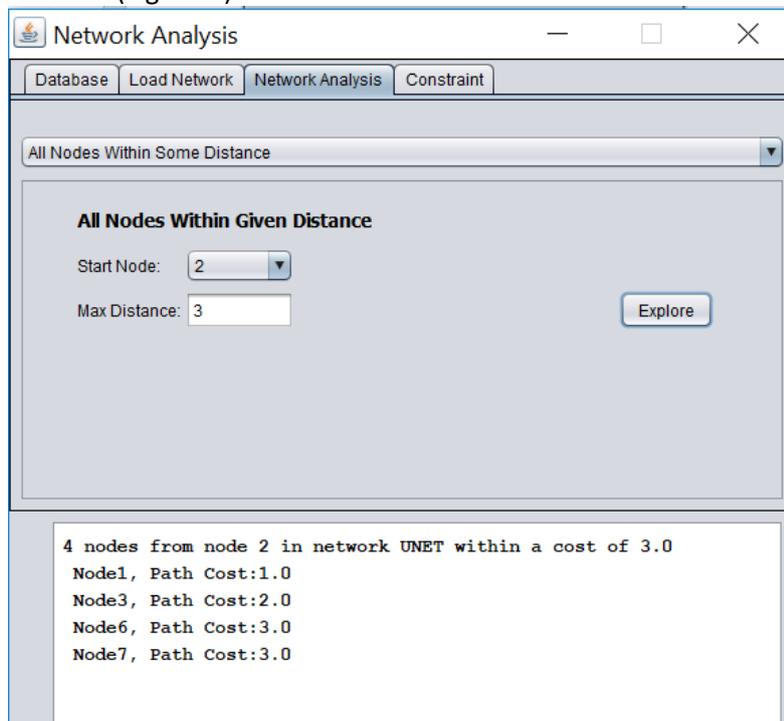


Figure-9 All nodes within distance analysis result

## CONSTRAINTS

The **SystemConstraint** class is a specific example of a network constraint. It lets you define a set of constraints to limit the search space for any of the methods you have seen so far.

- **MaxCost:** The maximum cost.
- **MaxDepth:** The maximum search depth (the number of links in the paths).
- **MaxDistance:** The maximum geographical distance from the start node and any candidate node (in other words, consider only those nodes within that distance from that start node)
- **MaxMBR:** Consider only those nodes that are inside the MBR.
- **MustAvoidLinks:** A list of links to avoid.
- **MustAvoidNodes:** A list of nodes to avoid.

Lets see how to set up a constraint that avoids nodes and limits the cost of any solution to “cost units.”

To create this panel add a new JPanel into the JTabbedPane and name the tab as “Constraint”. In this panel add objects . `jCheckBoxCost`, `jCheckBoxAvoidNode`, `jtxtCost`, `jTextAvoidedNodes`, `jButtonApplyConstraint` respectively as shown in Figure-10.

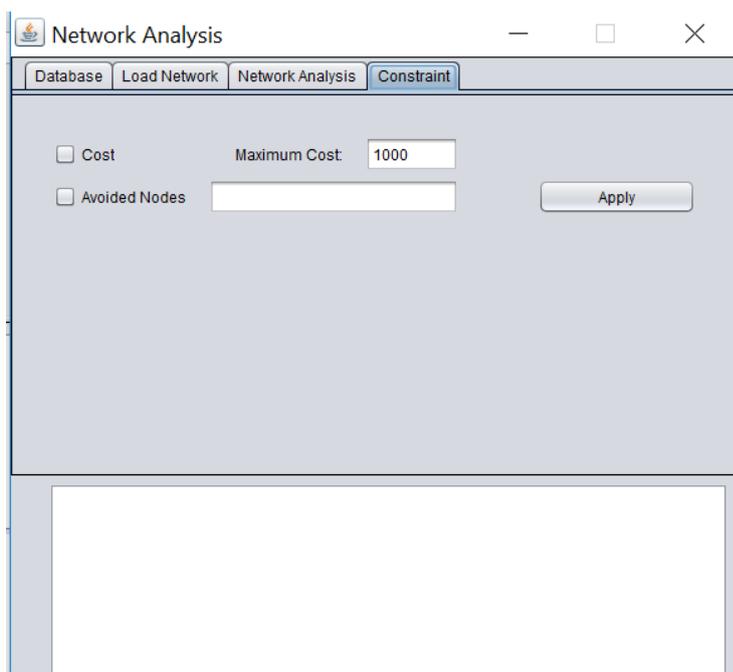


Figure-10 “Constraint” panel form design

Add Constraint object in the NetworkMain class as shown in Code Part-21.

```
public class NetworkMain extends javax.swing.JFrame {  
    private static Connection con;  
    private Network unet;  
    private SystemConstraint constraint;  
    private formShortestPath frmShortestPath;  
    private formNearestNeighbors frmNearestNeighbor;  
    private formNodesWithinDistance frmNodesWithinDistance;  
}
```

Code Part-21 Adding constraint object in the NetworkMain class.

Add action click method of “Apply Constraint” button and put required code as shown in Code Part-22.

```
private void jButtonApplyConstraintActionPerformed(java.awt.event.ActionEvent evt) {
    if(unet != null)
    {
        constraint = new SystemConstraint(unet);

        if(jCheckBoxCost.isSelected()){
            double cost = Double.parseDouble(jtxtCost.getText().trim());
            constraint.setMaxCost(cost);
        }
        if(jCheckBoxAvoidNode.isSelected())
        {
            Vector nodesVector = new Vector();
            String[] nodesArrayString = jTextAvoidedNodes.getText().split(",");
            for(String s:nodesArrayString)
            {
                try {
                    nodesVector.add(unet.getNode(Integer.parseInt(s)));
                } catch (NetworkDataException ex) {
                    NetworkMain.jTextOutput.setText(ex.getMessage());
                }
            }
            constraint.setMustAvoidNodes(nodesVector);
        }
        NetworkMain.jTextOutput.setText("Constraints applied...");
    }
    else
        NetworkMain.jTextOutput.setText("No network loaded...");
}
```

Code Part-22 “jButtonApplyConstraint” button action method.

Now let’s add required code in formShortestPath class to make constraints available as shown in CodePart-23.

```
public class formShortestPath extends javax.swing.JPanel {

    Network net;
    Connection cnn;
    Path path;
    SystemConstraint constraint;

    /**
     * Creates new form formShortestPath
     * @param net
     * @param cnn
     * @param constraint
     */
    public formShortestPath(Network net, Connection cnn, SystemConstraint constraint) {
        initComponents();
        this.net = net;
        this.cnn = cnn;
        this.constraint = constraint;
    }
}
```

Code Part-23 Changes on “formShortestPath” class and it’s constructor

Code Part-24 shows changes on findShortestPath method to apply constraints.

```
private void findShortestPath(int startNode, int endNode, SystemConstraint constraint)
{
    try {
        path = NetworkManager.shortestPath(net, startNode, endNode, constraint);
        if (path != null)
        {
            NetworkMain.jTextOutput.setText("Path Cost: " + path.getCost());
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Number of Links: " + path.getNoOfLinks());
            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Simple Path? " + path.isSimple());

            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\nLinks Traversed:");
            Link[] linkArray = path.getLinkArray();
            for (int i = 0; i < linkArray.length; i++) {
                NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Link " + linkArray[i].getID() + "\t"
                    + linkArray[i].getName() + "\t(cost:" + linkArray[i].getCost() + ")");
            }

            NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Nodes traversed:");
            Node[] nodeArray = path.getNodeArray();
            for (int i = 0; i < nodeArray.length; i++) {
                NetworkMain.jTextOutput.setText(NetworkMain.jTextOutput.getText() + "\n Node: " + nodeArray[i].getID() + "\t"
                    + nodeArray[i].getName() + "\t" + nodeArray[i].getCost());
            }
        } else {
            NetworkMain.jTextOutput.setText("Could not find solution...");
        }
    } catch (NetworkDataException ex) {
        NetworkMain.jTextOutput.setText(ex.getMessage());
    }
}
```

Code Part-24 Enabling constraints on findShortestPath method.

Code Part-25 shows changes on jButtonShortestPath button action method to apply constraints.

```
private void jButtonShortestPathActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int startNode = Integer.parseInt(jComboStartNode.getSelectedItem().toString());
    int endNode = Integer.parseInt(jComboEndNode.getSelectedItem().toString());

    findShortestPath(startNode, endNode, constraint);
}
}
```

Code Part-26 shows changes on “jComboAnalyseType” combobox action method to apply constraints.

```

private void jComboAnalyseTypeActionPerformed(java.awt.event.ActionEvent evt) {
    int option = jComboAnalyseType.getSelectedIndex();
    if(option == 0) //Show shortest path form in JscrollPane
    {
        frmShortestPath = new formShortestPath(unet, con, constraint);
        jScrollPaneAnalyse.setViewportView(frmShortestPath);
        frmShortestPath.setVisible(true);
    }
    else if(option == 1)
    {
        frmNearestNeighbor = new formNearestNeighbors(unet, con);
        jScrollPaneAnalyse.setViewportView(frmNearestNeighbor);
        frmNearestNeighbor.setVisible(true);
    }
    else if(option == 2)
    {
        frmNodesWithinDistance = new formNodesWithinDistance(unet, con);
        jScrollPaneAnalyse.setViewportView(frmNodesWithinDistance);
        frmNodesWithinDistance.setVisible(true);
    }
}

```

In the Constraint section we only applied constraints on formShortestPath sub form. Please also write codes in the same way to apply constraints in other two sub forms “formNodesWithinDistance” and “formNodesWithinDistance”.