附錄三　程式碼

GraphAlgorithm.java

```
package emnet.algorithm;

import java.util.Vector;
import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Iterator;
import emnet.thread.Center;
import emnet.thread.Roamer;
import emnet.thread.Detourist;
import emnet.thread.DetourManager;


public class GraphAlgorithm {
    public GraphAlgorithm() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static Graph copyGraph(Graph oldGraph){
        Vector oldNodeSet,oldEdgeSet;

        oldNodeSet=oldGraph.getNodeSet();
        oldEdgeSet=oldGraph.getEdgeSet();

        return copyGraph(oldNodeSet,oldEdgeSet);
    }

    public static Graph copyGraph(Vector oldNodeSet,Vector oldEdgeSet){
        Vector newNodeSet=new Vector(),newEdgeSet=new Vector();

        for(int i=0;i<oldNodeSet.size();i++){

            // new node setting:
            // 1. check label
```

GraphAlgorithm.java

```
                // 2. check supply/demand
                // 3. check visit
                Node tempOldNode=(Node)oldNodeSet.elementAt(i);
                Node tempNewNode=new Node(i);

                if(tempOldNode.isSupply()){
                    tempNewNode.setSupply();
                }else if(tempOldNode.isDemand()){
                    tempNewNode.setDemand();
                }

                if(tempOldNode.isVisited()){
                    tempNewNode.visit();
                }

                newNodeSet.addElement(tempNewNode);
            }

            for(int i=0;i<oldEdgeSet.size();i++){

                // new edge setting:
                // 1. check label
                // 2. check n1/n2
                // 3. set weight
                // 4. set previous node
                // 5. check fast
                // 6. check detour
                // 7. check visit
                Edge tempOldEdge=(Edge)oldEdgeSet.elementAt(i);
                int n1=tempOldEdge.getN1Label(),n2=tempOldEdge.getN2Label();
                double weight=tempOldEdge.getWeight();
                Edge tempNewEdge=new
Edge(i,(Node)newNodeSet.elementAt(n1),(Node)newNodeSet.elementAt(n2),weight);

                if(tempOldEdge.isFastEdge()){
                    tempNewEdge.setFastEdge();
                }

                if(tempOldEdge.isDetourEdge()){
                    tempNewEdge.setDetourEdge();
                }
```

GraphAlgorithm.java

```
            if(tempOldEdge.isVisited()){
                tempNewEdge.visit();
            }

            newEdgeSet.addElement(tempNewEdge);
        }
        return new Graph(newNodeSet,newEdgeSet);
    }

    public static Graph getSubtreeWithCertainNode(Graph tree,Node certainNode,Edge ruinedEdge){

        Vector nodeSet=new Vector(),edgeSet=new Vector();
        Vector currAdjacentNodes,currIncidentEdges;
        Node currNode=certainNode;

        int maxLabel=0;
        Vector tempTreeNodeSet=tree.getNodeSet();
        Node tempTreeNode;
        for(int i=0;i<tempTreeNodeSet.size();i++){
            tempTreeNode=(Node)tempTreeNodeSet.elementAt(i);
            if(tempTreeNode.getLabel()>maxLabel)
                maxLabel=tempTreeNode.getLabel();
        }

        Vector preNodes=new Vector(maxLabel+1);
        for(int i=0;i<(maxLabel+1);i++)
            preNodes.addElement(null);

        boolean finish=false;

        do{
            currAdjacentNodes=tree.adjacentNodeSet(currNode);
            currIncidentEdges=tree.incidentEdgeSet(currNode);

            if(currIncidentEdges.contains(ruinedEdge)){
                currIncidentEdges.removeElement(ruinedEdge);
                currAdjacentNodes.removeElement(ruinedEdge.theOtherNode(currNode));
            }

            Iterator itrAdjacentNodes=currAdjacentNodes.iterator();

            if(!nodeSet.contains(currNode))
```

GraphAlgorithm.java

```
                nodeSet.addElement(currNode);

            int i=0;
            currNodeAssign:
            do{
                if(itrAdjacentNodes.hasNext()){
                    Node tempNode=(Node)itrAdjacentNodes.next();

                    if(!nodeSet.contains(tempNode)){
                        preNodes.setElementAt(currNode,tempNode.getLabel());
                        currNode=tempNode;
                        break currNodeAssign;
                    }else{
                        i++;
                        if(i==currAdjacentNodes.size()){
                            if(currNode==certainNode){
                                finish=true;
                                break currNodeAssign;
                            }
                            Edge
tempEdge=tree.getEdge(currNode,(Node)preNodes.elementAt(currNode.getLabel()));
                            if(!edgeSet.contains(tempEdge)){
                                edgeSet.addElement(tempEdge);
                            }
                            currNode=(Node)preNodes.elementAt(currNode.getLabel());
                        }
                    }
                }else{
                    finish=true;
                    //break currNodeAssign;
                }
            }while(itrAdjacentNodes.hasNext());
        }while(!finish);
        return new Graph(nodeSet,edgeSet);
    }

    public static Vector getAdjacentNodes(Graph graph,Node currNode){
        return graph.adjacentNodeSet(currNode);
    }

    public static Vector getIncidnetEdges(Graph graph,Node currNode){
        return graph.incidentEdgeSet(currNode);
```

```
    }


    public static Graph getSubtreeWithoutSupply(Graph tree,Edge ruinedEdge){
        Graph subtreeWithoutSupply=null;
        Graph subtreeN1=getSubtreeWithCertainNode(tree,ruinedEdge.getN1(),ruinedEdge);
        Graph subtreeN2=getSubtreeWithCertainNode(tree,ruinedEdge.getN2(),ruinedEdge);
        Vector subtreeNodeSetN1=subtreeN1.getNodeSet();
        Node tempNode;
        for(int i=0;i<subtreeNodeSetN1.size();i++){
            tempNode=(Node)subtreeNodeSetN1.elementAt(i);
            if(tempNode.isSupply())
                subtreeWithoutSupply=subtreeN2;
        }
        if(subtreeWithoutSupply!=subtreeN2)
            subtreeWithoutSupply=subtreeN1;
        return subtreeWithoutSupply;
    }

    public static Graph getSubtreeWithSupply(Graph tree,Edge ruinedEdge){
        Graph subtreeWithSupply=null;
        Graph subtreeN1=getSubtreeWithCertainNode(tree,ruinedEdge.getN1(),ruinedEdge);
        Graph subtreeN2=getSubtreeWithCertainNode(tree,ruinedEdge.getN2(),ruinedEdge);
        Vector subtreeNodeSetN1=subtreeN1.getNodeSet();
        Node tempNode;
        for(int i=0;i<subtreeNodeSetN1.size();i++){
            tempNode=(Node)subtreeNodeSetN1.elementAt(i);
            if(tempNode.isSupply())
                subtreeWithSupply=subtreeN1;
        }
        if(subtreeWithSupply!=subtreeN1)
            subtreeWithSupply=subtreeN2;
        return subtreeWithSupply;
    }


    private void jbInit() throws Exception {
    }

    public static Vector getIncidentEdgeSet(Roamer roamer,Vector nodeSet){
        Center center=roamer.getCenter();
        Graph graph=center.getGraph();
```

GraphAlgorithm.java

```
        Edge myEdge=center.getMyEdge(roamer);
        Vector visitorSequence;

        Vector edgeSet=new Vector();
        Edge tempEdge;
        Node n1,n2;
        for(int i=0;i<nodeSet.size();i++){
            Vector tempEdgeSet=graph.incidentEdgeSet((Node)nodeSet.elementAt(i));
            for(int j=0;j<tempEdgeSet.size();j++){
                if(!edgeSet.contains(tempEdgeSet.elementAt(j))){
                    edgeSet.addElement(tempEdgeSet.elementAt(j));

                    tempEdge=(Edge)tempEdgeSet.elementAt(j);
                    n1=tempEdge.getN1();
                    n2=tempEdge.getN2();
                    //remove edges included in the routeEdgeSet
                    if(nodeSet.contains(n1) && nodeSet.contains(n2))
                        edgeSet.removeElement(tempEdge);
                    //remove dummyEdges
                    if(n1==center.getDummyNode() || n2==center.getDummyNode())
                        edgeSet.removeElement(tempEdge);
                    //remove visited edge
                    if(nodeSet.contains(n1)){
                        visitorSequence=center.getVisitorSequence(n2);
                        if(visitorSequence.contains(roamer))
                            edgeSet.removeElement(tempEdge);
                    }else if(nodeSet.contains(n2)){
                        visitorSequence=center.getVisitorSequence(n1);
                        if(visitorSequence.contains(roamer))
                            edgeSet.removeElement(tempEdge);
                    }
                }
            }
        }
        if(edgeSet.contains(myEdge))
            edgeSet.removeElement(myEdge);

        return edgeSet;
    }

    public static Vector getIncidentEdges(Detourist detourist,Vector nodeSet){
        Graph usableGraph=detourist.getUsableGraph();
```

GraphAlgorithm.java

```
        DetourManager dmr=detourist.getDetourManager();
        Center center=dmr.getCenter();
        Edge myEdge=dmr.getMyEdge(detourist);

        Vector incidentEdges=new Vector();
        Edge tempEdge;
        Node n1,n2;

        for(int i=0;i<nodeSet.size();i++){
            Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)nodeSet.elementAt(i));
            for(int j=0;j<tempEdgeSet.size();j++){
                tempEdge=(Edge)tempEdgeSet.elementAt(j);
                if(!incidentEdges.contains(tempEdge))
                    incidentEdges.addElement(tempEdge);

                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(nodeSet.contains(n1) && nodeSet.contains(n2))
                    incidentEdges.removeElement(tempEdge);
                if(n1==center.getDummyNode() || n2==center.getDummyNode())
                    incidentEdges.removeElement(tempEdge);
            }
        }
        if(incidentEdges.contains(myEdge))
            incidentEdges.removeElement(myEdge);

        return incidentEdges;
    }

    public static Vector getInterfaceNodes(Graph subject,Graph environment){
        Vector interfaceNodes=new Vector();
        Vector environmentEdgeSet=environment.getEdgeSet();

        Edge tempEdge;
        Node n1,n2;

        for(int i=0;i<environmentEdgeSet.size();i++){
            tempEdge=(Edge)environmentEdgeSet.elementAt(i);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();

            if(subject.hasNode(n1) && !subject.hasNode(n2)){
```

GraphAlgorithm.java

```
                if(!interfaceNodes.contains(n1))
                    interfaceNodes.addElement(n1);
            }
            if(subject.hasNode(n2) && !subject.hasNode(n1)){
                if(!interfaceNodes.contains(n2))
                    interfaceNodes.addElement(n2);
            }
        }
        return interfaceNodes;
    }


    public static double networkCost(Graph graph){
        Vector edgeSet=graph.getEdgeSet();
        Edge edge;
        double networkCost=0.0;
        for(int i=0;i<edgeSet.size();i++){
            edge=(Edge)edgeSet.elementAt(i);
            networkCost=networkCost+edge.getWeight();
        }
        return networkCost;
    }

    public static double pathCost(Vector edgeSet){
        double pathkCost=0.0;
        Edge edge;
        for(int i=0;i<edgeSet.size();i++){
            edge=(Edge)edgeSet.elementAt(i);
            pathkCost=pathkCost+edge.getWeight();
        }
        return pathkCost;
    }

    public static Vector intersection(Vector set0,Vector set1){
        Vector intersection=new Vector();
        Object temp;
        for(int i=0;i<set0.size();i++){
            temp=set0.elementAt(i);
            if(set1.contains(temp))
                intersection.addElement(temp);
        }
        return intersection;
```

GraphAlgorithm.java

```
}

public static Vector union(Vector set0,Vector set1){
    Vector union=new Vector();
    Object temp;
    for(int i=0;i<set0.size();i++){
        temp=set0.elementAt(i);
        union.addElement(temp);
    }
    for(int i=0;i<set1.size();i++){
        temp=set1.elementAt(i);
        if(!union.contains(temp))
            union.addElement(temp);
    }
    return union;
}

//detour
public static double getMergeCost(Detourist detourist,Node node){

    Center center=detourist.getDetourManager().getCenter();
    double mergeCost=0.0;

    Graph downstream=detourist.getDownstream();
    Vector downstreamDemandNodeSet=downstream.getDemandNodeSet();

    Graph mergeNodeToSupplyPath=getPathToSupply(node,center);
    Vector mergeNodeToSupplyEdgeSet=mergeNodeToSupplyPath.getEdgeSet();

    Vector tempEdgeSetUnion=new Vector(),tempEdgeSetIntersection=new Vector();

    Node tempDemand;
    Graph tempDemandToSupplyPath;
    Vector tempDemandToSupplyEdgeSet;
    for(int i=0;i<downstreamDemandNodeSet.size();i++){
        tempDemand=(Node)downstreamDemandNodeSet.elementAt(i);

        tempDemandToSupplyPath=getPathToSupply(tempDemand,center);
        tempDemandToSupplyEdgeSet=tempDemandToSupplyPath.getEdgeSet();

        tempEdgeSetUnion=union(mergeNodeToSupplyEdgeSet,tempDemandToSupplyEdgeSet);
```

GraphAlgorithm.java

```
tempEdgeSetIntersection=intersection(mergeNodeToSupplyEdgeSet,tempDemandToSupplyEdgeSet);

                mergeCost=mergeCost+pathCost(tempEdgeSetUnion)-pathCost(tempEdgeSetIntersection);
            }
            return mergeCost;
        }

        public static Graph getPathToSupply(Node node,Center center){
            Vector nodeSet=new Vector();
            Vector edgeSet=new Vector();

            Node currNode=node;
            Edge preEdge;
            while(currNode!=center.getDummyNode()){
                preEdge=center.getPreEdge(currNode);

                if(center.getPreNode(currNode)!=center.getDummyNode()){
                    if(!nodeSet.contains(currNode))
                        nodeSet.addElement(currNode);
                    if(!edgeSet.contains(preEdge))
                        edgeSet.addElement(preEdge);
                }else{
                    if(!nodeSet.contains(currNode))
                        nodeSet.addElement(currNode);
                }
                currNode=preEdge.theOtherNode(currNode);
            }
            return new Graph(nodeSet,edgeSet);
        }
}
```

Graph.java

```
package emnet.graph;

import java.util.Vector;

public class Graph {
	private Vector nodeSet,edgeSet;
	int supplyNodeNum,demandNodeNum;

	public Graph(Vector nodeSet, Vector edgeSet){
		this.nodeSet=nodeSet;
		this.edgeSet=edgeSet;
	}

	public Vector getNodeSet(){
		return this.nodeSet;
	}

	public Vector getEdgeSet(){
		return this.edgeSet;
	}

	//incident edges of node n
	public Vector incidentEdgeSet(Node node){
		Vector incidentEdgeSet=new Vector();
		for(int i=0;i<this.edgeSet.size();i++){
			Edge tempEdge=(Edge)edgeSet.elementAt(i);
			if(tempEdge.getN1()==node || tempEdge.getN2()==node){
				if(!incidentEdgeSet.contains(tempEdge)){
					incidentEdgeSet.addElement(tempEdge);
				}
			}
		}
		return incidentEdgeSet;
	}

	//adjacent nodes of node n
	public Vector adjacentNodeSet(Node n){
		Vector adjacentNodeSet=new Vector();
		for(int i=0;i<this.edgeSet.size();i++){
			Edge tempEdge=(Edge)edgeSet.elementAt(i);
			if(tempEdge.getN1()==n && !adjacentNodeSet.contains(tempEdge.getN2())){
```

Graph.java

```
                adjacentNodeSet.addElement(tempEdge.getN2());
            }else if(tempEdge.getN2()==n && !adjacentNodeSet.contains(tempEdge.getN1())){
                adjacentNodeSet.addElement(tempEdge.getN1());
            }
        }
        return adjacentNodeSet;
    }

    public Vector getSupplyNodeSet(){
        Vector supplyNodeSet=new Vector();
        for(int i=0;i<this.nodeSet.size();i++){
            Node tempNode=(Node)nodeSet.elementAt(i);
            if(tempNode.isSupply())
                supplyNodeSet.addElement(tempNode);
        }
        return supplyNodeSet;
    }

    public int getSupplyNodeNum(){
        return getSupplyNodeSet().size();
    }

    public Vector getDemandNodeSet(){
        Vector demandNodeSet=new Vector();
        for(int i=0;i<this.nodeSet.size();i++){
            Node tempNode=(Node)nodeSet.elementAt(i);
            if(tempNode.isDemand())
                demandNodeSet.addElement(tempNode);
        }
        return demandNodeSet;
    }

    public int getDemandNodeNum(){
        return getDemandNodeSet().size();
    }

    public Edge getEdge(Node n1,Node n2){
        Edge edge;
        for(int i=0;i<edgeSet.size();i++){
            edge=(Edge)edgeSet.elementAt(i);
            if(edge.getN1()==n1){
                if(edge.theOtherNode(edge.getN1())==n2)
```

Graph.java

```
                return edge;
            }else if(edge.getN1()==n2){
                if(edge.theOtherNode(edge.getN1())==n1)
                    return edge;
            }
        }
        System.out.println("error: no edge can be returned!");
        return null;
    }

    public void addNode(Node node){
        if(!nodeSet.contains(node))
            this.nodeSet.addElement(node);
    }

    public void addEdge(Edge edge){
        if(!edgeSet.contains(edge))
            this.edgeSet.addElement(edge);
    }

    public boolean hasNode(Node node){
        if(nodeSet.contains(node)){
            return true;
        }else{
            return false;
        }
    }

    public boolean hasEdge(Edge edge){
        if(edgeSet.contains(edge)){
            return true;
        }else{
            return false;
        }
    }

    public void removeNode(Node node){
        if(nodeSet.contains(node)){
            Vector incidentEdges=this.incidentEdgeSet(node);
            Edge tempIncidentEdge;
            for(int i=0;i<incidentEdges.size();i++){
                tempIncidentEdge=(Edge)incidentEdges.elementAt(i);
```

Graph.java

```
                    if(edgeSet.contains(tempIncidentEdge))
                        edgeSet.removeElement(tempIncidentEdge);
                }
                nodeSet.removeElement(node);
            }
        }

        public void removeEdge(Edge edge){
            if(edgeSet.contains(edge))
                edgeSet.removeElement(edge);
        }
    }
```

Node.java

```
package emnet.graph;

public class Node{
	private int label;
	private double x,y;
	private boolean demand,supply,merge,access,source,dummy;
	private boolean visit,occupy;

	public Node(int label){
		this.label=label;
		this.demand=false;
		this.supply=false;
		this.merge=false;
		this.access=false;
		this.source=false;
		this.dummy=false;
		this.visit=false;
		this.occupy=false;
	}

	public Node(int label,double x,double y){
		this(label);
		setX(x);
		setY(y);
	}

	public void setDemand(){
		this.demand=true;
	}

	public void setSupply(){
		this.supply=true;
	}

	public void setNeutral(){
		this.supply=false;
		this.demand=false;
	}

	public void setMerge(){
		this.merge=true;
```

Node.java

```
}

public void setAccess(){
    this.access=true;
}

public void setSource(){
    this.source=true;
}

public void setDummy(){
    this.dummy=true;
}

public synchronized void visit(){
    this.visit=true;
}

public synchronized void occupy(){
    this.occupy=true;
}

public synchronized void unOccupied(){
    this.occupy=false;
}

public synchronized void leave(){
    this.occupy=false;
}

public boolean isOccupied(){
    return this.occupy;
}

public boolean isDemand(){
    return this.demand;
}

public boolean isSupply(){
    return this.supply;
}
```

Node.java

```
	public boolean isMerge(){
		return this.merge;
	}

	public boolean isAccess(){
		return this.access;
	}

	public boolean isSource(){
		return this.source;
	}

	public boolean isDummy(){
		return this.dummy;
	}

	public boolean isVisited(){
		return this.visit;
	}

	public int getLabel(){
		return label;
	}

	public void setX(double x){
		this.x=x;
	}

	public void setY(double y){
		this.y=y;
	}

	public double getX(){
		return this.x;
	}

	public double getY(){
		return this.y;
	}
}
```

Edge.java

```
package emnet.graph;

public class Edge {
	private int label;
	private double weight;
	private Node n1,n2;
	private boolean fastEdge,detourEdge,testEdge,detourTestEdge,dummyEdge,maTestEdge,maEdge;
	private boolean visit;

	public Edge(int label,Node n1,Node n2){
		this.label=label;
		this.weight=0.0;
		this.n1=n1;
		this.n2=n2;
		this.fastEdge=false;
		this.detourEdge=false;
		this.testEdge=false;
		this.detourTestEdge=false;
		this.dummyEdge=false;
		this.maTestEdge=false;
		this.maEdge=false;
		this.visit=false;
	}

	public Edge(int label,Node n1,Node n2,double weight){
		this(label,n1,n2);
		this.weight=weight;
	}

	public synchronized void setFastEdge(){
		this.fastEdge=true;
	}

	public boolean isFastEdge(){
		return this.fastEdge;
	}

	public synchronized void setDetourEdge(){
		this.detourEdge=true;
	}
```

Edge.java

```
public boolean isDetourEdge(){
    return this.detourEdge;
}

public synchronized void setTestEdge(){
    this.testEdge=true;
}

public boolean isTestEdge(){
    return this.testEdge;
}

public synchronized void setDetourTestEdge(boolean detourTestEdge){
    this.detourTestEdge=detourTestEdge;
}

public boolean isDetourTestEdge(){
    return detourTestEdge;
}

public synchronized void setDummyEdge(){
    this.dummyEdge=true;
}

public boolean isDummyEdge(){
    return dummyEdge;
}

public synchronized void setMATestEdge(boolean maTestEdge){
    this.maTestEdge=maTestEdge;
}

public boolean isMATestEdge(){
    return maTestEdge;
}

public synchronized void setMAEdge(){
    this.maEdge=true;
}

public boolean isMAEdge(){
    return maEdge;
```

Edge.java

```
}

public synchronized void setNeutralEdge(){
    this.fastEdge=false;
    this.detourEdge=false;
    this.maEdge=false;
    this.testEdge=false;
    this.detourTestEdge=false;
    this.maTestEdge=false;
}

public void setWeight(double weight){
    this.weight=weight;
}

public double getWeight(){
    return this.weight;
}

public Node getN1(){
    return this.n1;
}

public Node getN2(){
    return this.n2;
}

public Node theOtherNode(Node n){
    if(n==this.n1){
        return this.n2;
    }else if(n==this.n2){
        return this.n1;
    }
    return null;
}

public synchronized void visit(){
    this.visit=true;
}

public boolean isVisited(){
    return this.visit;
```

Edge.java

```
        }

        public int getN1Label(){
            return n1.getLabel();
        }

        public int getN2Label(){
            return n2.getLabel();
        }

        public int getLabel(){
            return this.label;
        }
}
```

Map.java

```
package emnet.gui;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.util.Vector;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import emnet.Frame;
import java.text.DecimalFormat;
import java.awt.Font;

public class Map extends javax.swing.JPanel{
    Frame frame;
    JPanel map=new JPanel();
    Graph graph;
    boolean dataIn;
    boolean supply,demand,neutral;
    DecimalFormat myFormatter=new DecimalFormat("###,###.#");

    public Map(){
        init();
    }

    public void init(){
        this.setLayout(new BorderLayout());
        this.setSize(new Dimension(600,400));
        this.setPreferredSize(new Dimension(600,400));
        this.add(map,BorderLayout.CENTER);
        this.dataIn=false;

        this.addMouseListener(new Map_MouseAdapter(this));
    }

    public void paint(Graphics g){
        if(dataIn){
```

Map.java

```
                drawNodes(graph,g);
                drawEdges(graph,g);
                frame.setSeperator(graph);
            }else{
                g.drawString("n/a",this.getWidth()/2,this.getHeight()/2);
            }
        }

        public void setGraph(Graph graph){
            this.graph=graph;
            this.dataIn=true;
            this.repaint();
        }

        public Graph getGraph(){
            return this.graph;
        }
        public void sentFrame(Frame frame){
            this.frame=frame;
        }

        public Map getMap(){
            return this;
        }

        private void drawNodes(Graph graph,Graphics g){
            double ratio=scaledRatio(graph);
            double newOX=newOX(graph);
            double newOY=newOY(graph);
            int l=12,m=10,s=8;

            g.setFont(new Font(null,Font.PLAIN,12));

            Vector nodeSet=graph.getNodeSet();
            Node node;
            for(int i=0;i<nodeSet.size();i++){
                node=(Node)nodeSet.elementAt(i);

                if(node.isSupply()){
                    g.setColor(Color.RED);
                    g.drawOval(new Double((node.getX()-newOX)*ratio).intValue()-l/2,new
Double((node.getY()-newOY)*ratio).intValue()-l/2,l,l);
```

Map.java

```
                }else if(node.isDemand()){
                    g.setColor(Color.BLUE);
                    g.drawRect(new Double((node.getX()-newOX)*ratio).intValue()-m/2,new
Double((node.getY()-newOY)*ratio).intValue()-m/2,m,m);
                }else{
                    g.setColor(Color.LIGHT_GRAY);
                }
                g.drawString(""+node.getLabel(),new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue());

                if(node.isAccess() && node.isMerge()){
                    g.setColor(Color.DARK_GRAY);
                    g.drawString("A & M",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
                }else if(node.isAccess()){
                    g.setColor(Color.DARK_GRAY);
                    g.drawString("A",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
                }else if(node.isMerge()){
                    g.setColor(Color.DARK_GRAY);
                    g.drawString("M",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
                }

                if(node.isSource()){
                    g.setColor(Color.DARK_GRAY);
                    g.drawString("$rc",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
                }
            }
        }

        private void drawEdges(Graph graph,Graphics g){
            double ratio=scaledRatio(graph);
            double newOX=newOX(graph);
            double newOY=newOY(graph);

            Vector edgeSet=graph.getEdgeSet();
            Edge edge;

            g.setFont(new Font(null,Font.PLAIN,10));
```

Map.java

```
            for(int i=0;i<edgeSet.size();i++){
                edge=(Edge)edgeSet.elementAt(i);
                Node n1=edge.getN1(),n2=edge.getN2();
                int n1x=new Double((n1.getX()-newOX)*ratio).intValue();
                int n1y=new Double((n1.getY()-newOY)*ratio).intValue();
                int n2x=new Double((n2.getX()-newOX)*ratio).intValue();
                int n2y=new Double((n2.getY()-newOY)*ratio).intValue();

                if(edge.isFastEdge()){
                    g.setColor(Color.BLACK);
                }else if(edge.isDetourEdge()){
                    g.setColor(Color.ORANGE);
                }else if(edge.isMAEdge()){
                    g.setColor(Color.GREEN);
                }else if(edge.isTestEdge()){
                    g.setColor(Color.MAGENTA);
                }else if(edge.isDetourTestEdge()){
                    g.setColor(Color.CYAN);
                }else if(edge.isMATestEdge()){
                    g.setColor(Color.MAGENTA);
                }else{
                    g.setColor(Color.LIGHT_GRAY);
                }

                if(!edge.isDummyEdge()){
                    Double weight;
                    if(edge.isMAEdge()){
                        g.setFont(new Font(null,Font.BOLD,11));
                        g.drawLine(n1x,n1y,n2x,n2y);
                        weight=new Double(edge.getWeight());
                        g.drawString(""+edge.getLabel()+":"+myFormatter.format(weight)+" @B",new
Double((n1x+n2x)/2).intValue(),new Double((n1y+n2y)/2).intValue());
                        g.setFont(new Font(null,Font.PLAIN,10));
                    }else{
                        g.drawLine(n1x,n1y,n2x,n2y);
                        weight=new Double(edge.getWeight());
                        g.drawString(""+edge.getLabel()+":"+myFormatter.format(weight),new
Double((n1x+n2x)/2).intValue(),new Double((n1y+n2y)/2).intValue());
                    }
                }
            }
        }
```

Map.java

```
private double scaledRatio(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    double maxX=0.0,maxY=0.0;
    double newOX=newOX(graph),newOY=newOY(graph);
    Node node;
    for(int i=0;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if((node.getX()-newOX)>maxX){
            maxX=node.getX()-newOX;
        }else if((node.getY()-newOY)>maxY){
            maxY=node.getY()-newOY;
        }
    }
    return Math.min(map.getWidth()/maxX,map.getHeight()/maxY);
}

private double newOX(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    Node node=(Node)nodeSet.elementAt(0);
    double minX=node.getX();
    for(int i=1;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if(node.getX()<minX){
            minX=node.getX();
        }
    }
    return minX;
}

private double newOY(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    Node node=(Node)nodeSet.elementAt(0);
    double minY=node.getY();
    for(int i=1;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if(node.getY()<minY){
            minY=node.getY();
        }
    }
    return minY;
}
```

Map.java

```
    public void nodeSetting(boolean supply,boolean demand,boolean neutral){
        this.supply=supply;
        this.demand=demand;
        this.neutral=neutral;
    }

    void mouse_clicked_actionPerformed(MouseEvent e){
        double ratio=scaledRatio(graph);
        double newOX=newOX(graph),newOY=newOY(graph);
        Vector nodeSet=graph.getNodeSet();
        Node node;
        int err=10;
        for(int i=0;i<nodeSet.size();i++){
            node=(Node)nodeSet.elementAt(i);
            int x=new Double((node.getX()-newOX)*ratio).intValue();
            int y=new Double((node.getY()-newOY)*ratio).intValue();
            if(Math.abs(e.getX()-x)<err && Math.abs(e.getY()-y)<err){
                if(supply){
                    node.setNeutral();
                    node.setSupply();
                }
                if(demand){
                    node.setNeutral();
                    node.setDemand();
                }
                if(neutral){
                    node.setNeutral();
                }
            }
        }
    }
}

class Map_MouseAdapter extends MouseAdapter{
    Map adaptee;

    Map_MouseAdapter(Map adaptee){
        this.adaptee=adaptee;
    }

    public void mouseClicked(MouseEvent e){
```

Map.java

```
            this.adaptee.mouse_clicked_actionPerformed(e);
        }
    }
```

IOGraph.java

```
package emnet.io;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
import java.util.StringTokenizer;

public class IOGraph {

    private Graph g;

    public IOGraph(String dirName,String nodeFile,String edgeFile) throws IOException {
        File inputNodeFile=new File(dirName,nodeFile);
        File inputEdgeFile=new File(dirName,edgeFile);
        Vector nodeSet=new Vector();
        Vector edgeSet=new Vector();
        g=new Graph(nodeSet,edgeSet);

        //read node.txt
        FileReader nodeIn=new FileReader(inputNodeFile);
        BufferedReader buffNodeIn=new BufferedReader(nodeIn);

        String strLine;
        Node node;
        while((strLine=buffNodeIn.readLine())!=null){
            //delimiter of Tab is "\t"
            StringTokenizer strToken=new StringTokenizer(strLine,"\t",false);

            String[] nodeAttr=new String[3];
            for(int i=0;i<3;i++){
                nodeAttr[i]=strToken.nextToken();
            }
            node=new
Node(Integer.parseInt(nodeAttr[0]),Double.parseDouble(nodeAttr[1]),Double.parseDouble(nodeAttr[2]));
            nodeSet.addElement(node);
        }
```

IOGraph.java

```
            buffNodeIn.close();
            nodeIn.close();

            //read edge.txt
            FileReader edgeIn=new FileReader(inputEdgeFile);
            BufferedReader buffEdgeIn=new BufferedReader(edgeIn);

            Edge edge;
            while((strLine=buffEdgeIn.readLine())!=null){
                //delimiter of Tab is "\t"
                StringTokenizer strToken=new StringTokenizer(strLine,"\t",false);

                String[] edgeAttr=new String[3];
                for(int i=0;i<3;i++){
                    edgeAttr[i]=strToken.nextToken();
                }

                Node n1=(Node)nodeSet.elementAt(Integer.parseInt(edgeAttr[1]));
                Node n2=(Node)nodeSet.elementAt(Integer.parseInt(edgeAttr[2]));
                double
weight=Math.sqrt((n1.getX()-n2.getX())*(n1.getX()-n2.getX())+(n1.getY()-n2.getY())*(n1.getY()-n2.getY()));
                edge=new Edge(Integer.parseInt(edgeAttr[0]),n1,n2,weight);
                edgeSet.addElement(edge);
            }
            buffEdgeIn.close();
            edgeIn.close();
        }

        public Graph getGraph(){
            return this.g;
        }

}
```

Center.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import emnet.Frame;
import emnet.algorithm.GraphAlgorithm;

public class Center extends Thread{
	//Center
	Graph graph;
	Frame frame;
	boolean finish,available,detourCenterFinish,maCenterFinish;
	Node dummyNode;

	//nodeCenter
	int nodeNum;
	Object[][] nodeCenter;

	//roamerCenter
	int roamerNum;
	Object[][] roamerCenter;

	//fastCenter
	Object[][] fastCenter;

	//detourCenter
	int edgeNum,detourManagerNum;
	Object[][] detourCenter;

	//MutualAssistanceCenter
	Graph emnet;
	int maManagerNum;
	Object[][] maCenter;

	public Center(Graph graph,Frame frame){
		//Center
		this.graph=graph;
		this.frame=frame;
		finish=false;
```

Center.java

```
available=true;

//nodeCenter
//nodeCenter[node][0]: occupy(Boolean)
//nodeCenter[node][1]: distance(Double)
//nodeCenter[node][2]: preNode(Node)
//nodeCenter[node][3]: preEdge(Edge)
//nodeCenter[node][4]: visitorSequence(Vector)
//nodeCenter[node][5]: supply(Node)
//nodeCenter[node][6]: detourDist(Double)???

nodeNum=this.graph.getNodeSet().size();
nodeCenter=new Object[(nodeNum+1)][6];
dummyNode=new Node(nodeNum,0.0,0.0);

for(int i=0;i<(nodeNum+1);i++){
    nodeCenter[i][0]=new Boolean(false);
    nodeCenter[i][1]=new Double(0.0);
    nodeCenter[i][2]=null;
    nodeCenter[i][3]=null;
    nodeCenter[i][4]=new Vector();
    nodeCenter[i][5]=null;
}

//roamerCenter
//roamerCenter[roamer][0]: myNode(Node)
//roamerCenter[roamer][1]: myEdge(Edge)
//roamerCenter[roamer][2]: currNode(Node)
//roamerCenter[roamer][3]: routeSet(Graph)
roamerNum=graph.getSupplyNodeSet().size();
roamerCenter=new Object[roamerNum][4];
for(int i=0;i<roamerNum;i++){
    roamerCenter[i][0]=dummyNode;
    roamerCenter[i][1]=null;
    roamerCenter[i][2]=null;
    roamerCenter[i][3]=null;
}

//fastCenter
//fastCenter[demand][0]: fastPath(Graph)
//fastCenter[demand][1]: supply(Node)
//fastCenter[demand][2]: fastPathLength(Double)
```

Center.java

```
fastCenter=new Object[nodeNum][3];
for(int i=0;i<nodeNum;i++){
    fastCenter[i][0]=null;
    fastCenter[i][1]=null;
    fastCenter[i][2]=null;
}

//detourCenter
//detourCenter[edge][0]: downstream(Graph)
//detourCenter[edge][1]: upstream(Graph)
//detourCenter[edge][2]: mergeNode(Node)
//detourCenter[edge][3]: accessNode(Node)
//detourCenter[edge][4]: detourPath(Graph)
//detourCenter[edge][5]: systematicDetourCost(Double)
//detourCenter[edge][6]: mergeCost(Vector)
edgeNum=graph.getEdgeSet().size();
detourManagerNum=graph.getSupplyNodeSet().size();
detourCenter=new Object[edgeNum][7];
for(int i=0;i<edgeNum;i++){
    detourCenter[i][0]=null;
    detourCenter[i][1]=null;
    detourCenter[i][2]=null;
    detourCenter[i][3]=null;
    detourCenter[i][4]=null;
    detourCenter[i][5]=new Double(0.0);
    detourCenter[i][6]=new Vector(nodeNum);
}

//mutualAssistantCenter
//maCenter[supply][0]: fastTree(Graph)
//maCenter[supply][1]: territory(Graph) 2ECON
//maCenter[supply][2]: source(Node)
//maCenter[supply][3]: icpSet(Vector)
//maCenter[supply][4]: maPath(Graph)
//maCenter[supply][5]: within territory supply-demand ratio(Double)
//maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
//maCenter[supply][7]: maCost > source to supply
maManagerNum=graph.getSupplyNodeNum();
maCenter=new Object[nodeNum][8];
for(int i=0;i<nodeNum;i++){
    maCenter[i][0]=null;
    maCenter[i][1]=null;
```

Center.java

```
                maCenter[i][2]=null;
                maCenter[i][3]=new Vector();
                maCenter[i][4]=null;
                maCenter[i][5]=new Vector();
                maCenter[i][6]=new Double(0.0);
                maCenter[i][7]=new Double(0.0);
        }
    }

    public void run(){
        startRoamerCenter();
        startFastCenter();
        startDetourCenter();
        startMutualAssistanceCenter();
        startOutputReport();
    }

    //center method
    void startRoamerCenter(){
        //roamerCenter
        //send roamers to find fast paths (Shortest Path Forest, SPF)
        sendRoamer();

        Vector nodeSet=graph.getNodeSet();
        int demandNodeNum=graph.getDemandNodeNum();
        do{
            int totalTimes=0;
            watching:
                for(int i=0;i<nodeNum;i++){
                Node tempNode=(Node)nodeSet.elementAt(i);
                //finish condition is focused on demand nodes only
                if(tempNode.isDemand()){
                    if(getVisitorNum(tempNode)==roamerNum){
                        totalTimes=totalTimes+roamerNum;
                        if(totalTimes==(demandNodeNum*roamerNum))
                            finish=true;
                    }
                    else{
                        finish=false;
                        break watching;
                    }
                }
```

Center.java

```
                }
            }while(!finish);
            //roamer center finished!
        }

        void startFastCenter(){
            //fastCenter
            //fastCenter[demand][0]: fastPath(Graph)
            //fastCenter[demand][1]: supply(Node)
            //fastCenter[demand][2]: fastPathLength(Double)

            //finding fast paths from demand nodes to the dummyNode
            try{
                sleep(1);
            }catch(InterruptedException ex){
                //fast center cannot sleep!
            }

            Vector demandNodeSet=graph.getDemandNodeSet();
            Node tempDemand,tempNode;
            Edge tempPreEdge,tempEdge;
            double length;
            Vector tempFastNodeSet,tempFastEdgeSet;

            for(int i=0;i<demandNodeSet.size();i++){
                tempDemand=(Node)demandNodeSet.elementAt(i);
                tempNode=tempDemand;

                tempFastNodeSet=new Vector();
                tempFastEdgeSet=new Vector();

                find:
                while(tempNode!=dummyNode){
                    tempPreEdge=(Edge)getPreEdge(tempNode);
                    if(tempNode.isSupply()){
                        fastCenter[tempDemand.getLabel()][1]=tempNode;
                        if(!tempFastNodeSet.contains(tempNode))
                            tempFastNodeSet.addElement(tempNode);

                        //node of fast route belong to the same supply
                        Node tempNode2;
                        for(int j=0;j<tempFastNodeSet.size();j++){
```

Center.java

```
                    tempNode2=(Node)tempFastNodeSet.elementAt(j);
                    nodeCenter[tempNode2.getLabel()][5]=tempNode;
                }
                break find;
            }else{
                if(!tempFastNodeSet.contains(tempNode))
                    tempFastNodeSet.addElement(tempNode);
                if(!tempFastEdgeSet.contains(tempPreEdge))
                    tempFastEdgeSet.addElement(tempPreEdge);
                tempPreEdge.setFastEdge();
            }
            tempNode=tempPreEdge.theOtherNode(tempNode);
        }

        fastCenter[tempDemand.getLabel()][0]=new Graph(tempFastNodeSet,tempFastEdgeSet);
        length=0.0;
        for(int j=0;j<tempFastEdgeSet.size();j++){
            tempEdge=(Edge)tempFastEdgeSet.elementAt(j);
            length=length+tempEdge.getWeight();
        }
        fastCenter[tempDemand.getLabel()][2]=new Double(length);
    }

    //maCenter[supply][0]: fastTree(Graph)
    Vector fastTreeSupplyNodeSet=graph.getSupplyNodeSet();
    Node tempFastTreeSupply;
    Vector fastTreeDemandNodeSet=graph.getDemandNodeSet();
    Node tempFastTreeDemand;

    Graph tempFastPath;
    Vector tempFastPathNodeSet;
    Node tempFastPathNode;
    Vector tempFastPathEdgeSet;
    Edge tempFastPathEdge;

    Vector tempFastTreeNodeSet;
    Vector tempFastTreeEdgeSet;

    for(int i=0;i<fastTreeSupplyNodeSet.size();i++){
        tempFastTreeNodeSet=new Vector();
        tempFastTreeEdgeSet=new Vector();
```

Center.java

```
                tempFastTreeSupply=(Node)fastTreeSupplyNodeSet.elementAt(i);
                if(!tempFastTreeNodeSet.contains(tempFastTreeSupply))
                    tempFastTreeNodeSet.addElement(tempFastTreeSupply);

                for(int j=0;j<fastTreeDemandNodeSet.size();j++){
                    tempFastTreeDemand=(Node)fastTreeDemandNodeSet.elementAt(j);

                    //fast paths with different demands of the same supply
                    if(fastCenter[tempFastTreeDemand.getLabel()][1]==tempFastTreeSupply){
                        tempFastPath=(Graph)fastCenter[tempFastTreeDemand.getLabel()][0];
                        tempFastPathEdgeSet=tempFastPath.getEdgeSet();
                        for(int k=0;k<tempFastPathEdgeSet.size();k++){
                            tempFastPathEdge=(Edge)tempFastPathEdgeSet.elementAt(k);
                            if(!tempFastTreeEdgeSet.contains(tempFastPathEdge)){
                                tempFastTreeEdgeSet.addElement(tempFastPathEdge);
                            }
                        }

                        tempFastPathNodeSet=tempFastPath.getNodeSet();
                        for(int k=0;k<tempFastPathNodeSet.size();k++){
                            tempFastPathNode=(Node)tempFastPathNodeSet.elementAt(k);
                            if(!tempFastTreeNodeSet.contains(tempFastPathNode)){
                                tempFastTreeNodeSet.addElement(tempFastPathNode);
                            }
                        }
                    }
                }
                maCenter[tempFastTreeSupply.getLabel()][0]=new
Graph(tempFastTreeNodeSet,tempFastTreeEdgeSet);

                //maCenter[supply][5]: within territory supply-demand ratio(Double)
                if(getFastTree(tempFastTreeSupply).getDemandNodeNum()!=0){

this.setTerritorySDR(tempFastTreeSupply,1.0/getFastTree(tempFastTreeSupply).getDemandNodeNum());
                }else{
                    this.setTerritorySDR(tempFastTreeSupply,1.0);
                }

            }

            //fast center finished!
            //finalization
```

Center.java

```
        Vector edgeSet=graph.getEdgeSet();
        Edge clearEdge;
        for(int i=0;i<edgeSet.size();i++){
            clearEdge=(Edge)edgeSet.elementAt(i);
            if(!clearEdge.isFastEdge())
                clearEdge.setNeutralEdge();
        }
    }

    void startDetourCenter(){
        //detourCenter
        try{
            sleep(1);
        }catch(InterruptedException ex){
            //detour center cannot sleep!
        }

        detourCenterFinish=false;
        Vector supplyNodeSet=graph.getSupplyNodeSet();

        Node tempSupply;
        for(int i=0;i<supplyNodeSet.size();i++){
            tempSupply=(Node)supplyNodeSet.elementAt(i);
            new DetourManager(i,this,tempSupply).start();
        }

        while(!detourCenterFinish){
            //detour center waiting for detour managers finish their jobs
        }
        //detour center finished!
    }

    void startMutualAssistanceCenter(){
        try{
            sleep(1);
        }catch(InterruptedException ex){
            //mutual assistance center cannot sleep!
        }

        //where are supply nodes:
        Vector nodeSet=graph.getNodeSet();
        Node tempNode;
```

Center.java

```
for(int i=0;i<nodeSet.size();i++){
    tempNode=(Node)nodeSet.elementAt(i);
}

//initialization
Vector edgeSet=graph.getEdgeSet();
Edge tempEdge;
for(int i=0;i<edgeSet.size();i++){
    tempEdge=(Edge)edgeSet.elementAt(i);
    if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge())
        tempEdge.setNeutralEdge();
}

maCenterFinish=false;
Vector supplyNodeSet=graph.getSupplyNodeSet();

Node tempSupply;
for(int i=0;i<supplyNodeSet.size();i++){
    tempSupply=(Node)supplyNodeSet.elementAt(i);
    new MAManager(i,this,tempSupply).start();
}

while(!maCenterFinish){
    //ma center waiting for detour managers finish their jobs
}

//emnet
Vector emnetNodeSet=new Vector();
Vector emnetEdgeSet=new Vector();

Graph tempTerritory;
Vector tempTerritoryNodeSet;
Vector tempTerritoryEdgeSet;

Graph tempMAPath;
Vector tempMAPathNodeSet;
Vector tempMAPathEdgeSet;

for(int i=0;i<supplyNodeSet.size();i++){
    tempSupply=(Node)supplyNodeSet.elementAt(i);

    tempTerritory=this.getTerritory(tempSupply);
```

Center.java

```
            tempTerritoryNodeSet=tempTerritory.getNodeSet();
            tempTerritoryEdgeSet=tempTerritory.getEdgeSet();

            emnetNodeSet=GraphAlgorithm.union(emnetNodeSet,tempTerritoryNodeSet);
            emnetEdgeSet=GraphAlgorithm.union(emnetEdgeSet,tempTerritoryEdgeSet);

            tempMAPath=this.getMAPath(tempSupply);
            tempMAPathNodeSet=tempMAPath.getNodeSet();
            tempMAPathEdgeSet=tempMAPath.getEdgeSet();

            emnetNodeSet=GraphAlgorithm.union(emnetNodeSet,tempMAPathNodeSet);
            emnetEdgeSet=GraphAlgorithm.union(emnetEdgeSet,tempMAPathEdgeSet);
        }
        emnet=new Graph(emnetNodeSet,emnetEdgeSet);

        //ma center finished!
        //finalization
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
                tempEdge.setNeutralEdge();
        }
        //ma center closed!
    }

    void startOutputReport(){
        //maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
        double ld=0.0;
        Vector supplyNodeSet=graph.getSupplyNodeSet();
        Node tempSupply;
        Vector tempFastTreeEdgeSet;
        Edge tempRuinedEdge;
        for(int i=0;i<supplyNodeSet.size();i++){
            tempSupply=(Node)supplyNodeSet.elementAt(i);
            tempFastTreeEdgeSet=this.getFastTree(tempSupply).getEdgeSet();
            for(int j=0;j<tempFastTreeEdgeSet.size();j++){
                tempRuinedEdge=(Edge)tempFastTreeEdgeSet.elementAt(j);
                if(this.getSystematicDetourCost(tempRuinedEdge)>ld)
                    ld=this.getSystematicDetourCost(tempRuinedEdge);
            }
        }
        frame.setLD(ld);
```

Center.java

```
        double mac=0.0;
        for(int i=0;i<supplyNodeSet.size();i++){
            tempSupply=(Node)supplyNodeSet.elementAt(i);
            mac=mac+this.getMACost(tempSupply);
        }
        frame.setAMAC(mac/supplyNodeSet.size());
        frame.setNC(GraphAlgorithm.networkCost(emnet));

        double fastCost=0.0;
        double maxFastCost=0.0;
        Vector demandNodeSet=graph.getDemandNodeSet();
        Node tempDemand;
        for(int i=0;i<demandNodeSet.size();i++){
            tempDemand=(Node)demandNodeSet.elementAt(i);

            if(getFastCost(tempDemand)>maxFastCost)
                maxFastCost=getFastCost(tempDemand);

            fastCost=fastCost+this.getFastCost(tempDemand);
        }
        frame.setATC(fastCost/demandNodeSet.size());
        frame.setMTC(maxFastCost);
    }

    //center field:
    public synchronized Graph getGraph(){
        return this.graph;
    }

    public synchronized boolean isFinished(){
        return this.finish;
    }

    void sendRoamer(){
        Node supply;
        for(int i=0;i<this.graph.getSupplyNodeSet().size();i++){
            supply=(Node)this.graph.getSupplyNodeSet().elementAt(i);
            roamerCenter[i][0]=supply;
            new Roamer(i,supply,this).start();
        }
    }
```

Center.java

```
public Node getDummyNode(){
    return this.dummyNode;
}

public synchronized void takeKey(Roamer roamer){
    if(!finish){
        while(!available){
            try{
                //roamer is waiting to take!
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
        //roamer took the key!
        available=false;
    }else{
        available=false;
        //roamer took the key, but center is finished!
    }
}

public synchronized void putKey(Roamer roamer){
    if(!finish){
        while(available){
            try{
                //roamer is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //roamer put the key!
        available=true;
    }else{
        //roamer put the key & center is already finished!
        available=true;
    }
}

//detour manager take key
```

Center.java

```
public synchronized void takeKey(DetourManager dmr){
    if(!detourCenterFinish){
        while(!available){
            try{
                //dmr is waiting to take!
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
        //dmr took the key!
        available=false;
    }else{
        available=false;
        //dmr took the key, but detour center is finished!
    }
}

public synchronized void putKey(DetourManager dmr){
    if(!detourCenterFinish){
        while(available){
            try{
                //dmr is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //dmr put the key!
        available=true;
    }else{
        //dmr put the key! detour center is already finished!
        available=true;
    }
}

//ma manager take key
public synchronized void takeKey(MAManager maMr){
    if(!maCenterFinish){
        while(!available){
            try{
                //maMr is waiting to take!
```

Center.java

```
                    wait(1);
                }catch(InterruptedException e){
                    //takeKey: cannot wait!
                }
            }
            //maMr took the key!
            available=false;
        }else{
            available=false;
            //maMr took the key but maCenter is already finished!
        }
    }

    public synchronized void putKey(MAManager maMr){
        if(!maCenterFinish){
            while(available){
                try{
                    //maMr is waiting to put...
                    wait(1);
                }catch(InterruptedException e){
                    //putKey: cannot wait!
                }
            }
            //maMr put the key!
            available=true;
        }else{
            //maMr put the key! and maCenter already finished!
            available=true;
        }
    }

    public void updateDetourCondition(){
        detourManagerNum--;
        if(detourManagerNum==0)
            detourCenterFinish=true;
    }

    public void updateMACondition(){
        maManagerNum--;
        if(maManagerNum==0)
            maCenterFinish=true;
    }
```

Center.java

```
//nodeCenter method:
//nodeCenter[node][0]: occupy(Boolean)
public synchronized void setOccupy(Node node,boolean occupy){
    nodeCenter[node.getLabel()][0]=new Boolean(occupy);
    notifyAll();
}

//nodeCenter[node][1]: distance(Double)
public synchronized void setDistance(Node node,double distance){
    nodeCenter[node.getLabel()][1]=new Double(distance);
}

public synchronized double getDistance(Node node){
    Double distance=(Double)nodeCenter[node.getLabel()][1];
    return distance.doubleValue();
}

//nodeCenter[node][2]: preNode(Node)
public synchronized void setPreNode(Node node,Node preNode){
    nodeCenter[node.getLabel()][2]=preNode;
}

public synchronized Node getPreNode(Node node){
    Node preNode=(Node)nodeCenter[node.getLabel()][2];
    return preNode;
}

//nodeCenter[node][3]: preEdge(Edge)
public synchronized void setPreEdge(Node node,Edge preEdge){
    nodeCenter[node.getLabel()][3]=preEdge;
}

public synchronized Edge getPreEdge(Node node){
    Edge preEdge=(Edge)nodeCenter[node.getLabel()][3];
    return preEdge;
}

//nodeCenter[node][4]: visitorSequence(Vector)
public synchronized void addViditor(Node node,Roamer roamer){
    Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
    if(!visitorSequence.contains(roamer))
```

Center.java

```
            visitorSequence.addElement(roamer);
        nodeCenter[node.getLabel()][4]=visitorSequence;
    }

    public synchronized Vector getVisitorSequence(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        return visitorSequence;
    }

    public synchronized int getVisitorNum(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        int visitorNum=visitorSequence.size();
        return visitorNum;
    }

    public synchronized Roamer lastVisitor(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        Roamer lastVisitor=(Roamer)visitorSequence.lastElement();
        return lastVisitor;
    }

    //nodeCenter[node][5]: supply(Node)
    public Node getSupply(Node node){
        return (Node)nodeCenter[node.getLabel()][5];
    }

    //roamerCenter method:
    //roamerCenter[roamer][0]: myNode(Node)
    public synchronized void setMyNode(Roamer roamer,Node myNode){
        roamerCenter[roamer.getID()][0]=myNode;
    }

    public synchronized Node getMyNode(Roamer roamer){
        Node myNode=(Node)roamerCenter[roamer.getID()][0];
        return myNode;
    }

    //roamerCenter[roamer][1]: myEdge(Edge)
    public synchronized void setMyEdge(Roamer roamer,Edge myEdge){
        roamerCenter[roamer.getID()][1]=myEdge;
    }
```

Center.java

```
public synchronized Edge getMyEdge(Roamer roamer){
    Edge myEdge=(Edge)roamerCenter[roamer.getID()][1];
    return myEdge;
}

//roamerCenter[roamer][2]: currNode(Node)
public synchronized void setCurrNode(Roamer roamer,Node currNode){
    roamerCenter[roamer.getID()][2]=currNode;
}

public synchronized Node getCurrNode(Roamer roamer){
    Node currNode=(Node)roamerCenter[roamer.getID()][2];
    return currNode;
}

//roamerCenter[roamer][3]: routeSet(Graph)
public synchronized void setRouteSet(Roamer roamer,Graph routeSet){
    roamerCenter[roamer.getID()][3]=routeSet;
}

public synchronized void setRouteSet(Roamer roamer,Vector routeNodeSet,Vector routeEdgeSet){
    roamerCenter[roamer.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
}

public synchronized Graph getRouteSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    return routeSet;
}

public synchronized Vector getRouteNodeSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    Vector routeNodeSet=routeSet.getNodeSet();
    return routeNodeSet;
}

public synchronized Vector getRouteEdgeSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    Vector routeEdgeSet=routeSet.getEdgeSet();
    return routeEdgeSet;
}

public synchronized void addEdge(Roamer roamer,Edge edge){
```

Center.java

```
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Roamer roamer,Node node){
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeEdge(Roamer roamer,Edge edge){
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(routeEdgeSet.contains(edge)){
            routeEdgeSet.removeElement(edge);
        }
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeSubtree(Roamer roamer,Graph subtree){
        Graph routeSet=(Graph)this.roamerCenter[roamer.getID()][3];
        Vector tempNodeSet1=routeSet.getNodeSet();
        Vector tempEdgeSet1=routeSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            boolean nodeExist=tempNodeSet1.removeElement(tempNode);
        }

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
```

Center.java

```
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            boolean edgeExist=tempEdgeSet1.removeElement(tempEdge);
        }
    }

    public synchronized void addSubtree(Roamer roamer,Graph subtree){
        Graph visitedRouteSet=(Graph)this.roamerCenter[roamer.getID()][3];
        Vector tempNodeSet1=visitedRouteSet.getNodeSet();
        Vector tempEdgeSet1=visitedRouteSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();
        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            if(tempNodeSet1.contains(tempNode)){
            }else{
                tempNodeSet1.addElement(tempNode);
            }
        }

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            if(tempEdgeSet1.contains(tempEdge)){
            }else{
                tempEdgeSet1.addElement(tempEdge);
            }
        }
    }

    //fastCenter
    //fastCenter[demand][0]: fastPath(Graph)
    //fastCenter[demand][1]: supply(Node)
    //fastCenter[demand][2]: fastPathLength(Double)
    public double getFastCost(Node demand){
        Double fastCost=(Double)fastCenter[demand.getLabel()][2];
        return fastCost.doubleValue();
    }

    //detourCenter[edge][0]: downstream(Graph)
    public void setDownstream(Edge ruinedEdge,Graph downstream){
        detourCenter[ruinedEdge.getLabel()][0]=downstream;
```

Center.java

```
    }

    public Graph getDownstream(Edge ruinedEdge){
        return (Graph)detourCenter[ruinedEdge.getLabel()][0];
    }

    //detourCenter[edge][1]: upstream(Graph)
    public void setUpstream(Edge ruinedEdge,Graph upstream){
        detourCenter[ruinedEdge.getLabel()][1]=upstream;
    }

    public Graph getUpstream(Edge ruinedEdge){
        return (Graph)detourCenter[ruinedEdge.getLabel()][1];
    }

    //detourCenter[edge][2]: mergeNode(Node)
    public void setMergeNode(Edge ruinedEdge,Node mergeNode){
        detourCenter[ruinedEdge.getLabel()][2]=mergeNode;
    }

    public Node getMergeNode(Edge ruinedEdge){
        return (Node)detourCenter[ruinedEdge.getLabel()][2];
    }

    //detourCenter[edge][3]: accessNode(Node)
    public void setAccessNode(Edge ruinedEdge,Node accessNode){
        detourCenter[ruinedEdge.getLabel()][3]=accessNode;
    }

    public Node getAccessNode(Edge ruinedEdge){
        return (Node)detourCenter[ruinedEdge.getLabel()][3];
    }

    //detourCenter[edge][4]: detourPath(Graph)
    public void setDetourPath(Edge ruinedEdge,Graph detourPath){
        detourCenter[ruinedEdge.getLabel()][4]=detourPath;
    }

    public Graph getDetourPath(Edge ruinedEdge){
        return (Graph)detourCenter[ruinedEdge.getLabel()][4];
    }
```

Center.java

```
//detourCenter[edge][5]: systematicDetourCost(Double)
public void setSystematicDetourCost(Edge ruinedEdge,double sdc){
    detourCenter[ruinedEdge.getLabel()][5]=new Double(sdc);
}

public double getSystematicDetourCost(Edge ruinedEdge){
    Double tempDouble=(Double)detourCenter[ruinedEdge.getLabel()][5];
    return tempDouble.doubleValue();
}

//detourCenter[edge][6]: mergeCost(Vector)
public void setMergeCost(Edge ruinedEdge,double cost){
    detourCenter[ruinedEdge.getLabel()][6]=new Double(cost);
}

public double getMergeCost(Edge ruinedEdge){
    Double mergeCost=(Double)detourCenter[ruinedEdge.getLabel()][6];
    return mergeCost.doubleValue();
}

//maCenter
//maCenter[supply][0]: fastTree(Graph)
public Graph getFastTree(Node supply){
    return (Graph)maCenter[supply.getLabel()][0];
}

//maCenter[supply][1]: territory(Graph) 2ECON
public synchronized void setTerritory(Node supply,Graph territory){
    maCenter[supply.getLabel()][1]=territory;
}

public Graph getTerritory(Node supply){
    return (Graph)maCenter[supply.getLabel()][1];
}

//maCenter[supply][2]: source(Node)
public synchronized void setSource(Node supply,Node source){
    maCenter[supply.getLabel()][2]=source;
}

public Node getSource(Node supply){
    return (Node)maCenter[supply.getLabel()][2];
```

Center.java

```
}

//maCenter[supply][3]: icpSet(Vector)
public synchronized void setICPSet(Node supply,Vector icpSet){
    maCenter[supply.getLabel()][3]=icpSet;
}

public Vector getICPSet(Node supply){
    return (Vector)maCenter[supply.getLabel()][3];
}

public int getICPNum(Node supply){
    Vector icpSet=(Vector)maCenter[supply.getLabel()][3];
    return icpSet.size();
}

//maCenter[supply][4]: maPath(Graph)
public synchronized void setMAPath(Node supply,Graph maPath){
    maCenter[supply.getLabel()][4]=maPath;
}

public Graph getMAPath(Node supply){
    return (Graph)maCenter[supply.getLabel()][4];
}

//maCenter[supply][5]: within territory supply-demand ratio(Double)
public synchronized void setTerritorySDR(Node supply,double territorySDR){
    maCenter[supply.getLabel()][5]=new Double(territorySDR);
}

public double getTerritorySDR(Node supply){
    Double territorySDR=(Double)maCenter[supply.getLabel()][5];
    return territorySDR.doubleValue();
}

//maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
public synchronized void setMAsdr(Node supply,double maSDR){
    maCenter[supply.getLabel()][6]=new Double(maSDR);
}

public double getMAsdr(Node supply){
    Double maSDR=(Double)maCenter[supply.getLabel()][6];
```

Center.java

```
            return maSDR.doubleValue();
        }

        //maCenter[supply][7]: maCost > source to supply
        //including maPath cost & merge cost with respect to territory demandNum
        public synchronized void setMACost(Node supply,double maCost){
            maCenter[supply.getLabel()][7]=new Double(maCost);
        }

        public double getMACost(Node supply){
            Double maCost=(Double)maCenter[supply.getLabel()][7];
            return maCost.doubleValue();
        }
}
```

Detourist.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Graph;
import java.util.Vector;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class Detourist extends Thread{
    int id;
    Node start,supply,access;
    DetourManager dmr;
    Center center;

    Node currNode,preNode,myNode;
    Edge preEdge,myEdge,currRuinedEdge;
    Graph usableGraph,downstream,upstream,bridge,routeSet,detourPath;
    Vector routeNodeSet,routeEdgeSet,upstreamNodeSet;

    int downstreamDemandNum;
    double mergeCost,systematicDetourCost;

    boolean detouristFinish;



    public Detourist(int id,Node start,DetourManager dmr){
        super(""+id);
        this.id=id;
        this.start=start;
        this.dmr=dmr;
        center=dmr.getCenter();
        supply=dmr.getSupply();

        int edgeNum=dmr.getGraph().getEdgeSet().size();
        Node dummyNode=this.center.getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        dmr.getGraph().addNode(dummyNode);
```

Detourist.java

```
        dmr.getGraph().addEdge(dummyEdge);

        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);
        dmr.setRouteSet(this,routeNodeSet,routeEdgeSet);

        dmr.setMyNode(this,dummyNode);
        dmr.setMyEdge(this,dummyEdge);
        dmr.setCurrNode(this,start);

        Vector nodeSet=dmr.getGraph().getNodeSet();
        Vector edgeSet=dmr.getGraph().getEdgeSet();
        Vector usableEdgeSet=new Vector();
        Edge tempEdge;
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(tempEdge!=dmr.getCurrRuinedEdge())
                usableEdgeSet.addElement(tempEdge);
        }
        usableGraph=new Graph(nodeSet,usableEdgeSet);

        downstream=dmr.getCurrDownstream();
        upstream=dmr.getCurrUpstream();
        upstreamNodeSet=upstream.getNodeSet();

        currRuinedEdge=dmr.getCurrRuinedEdge();
        downstreamDemandNum=dmr.getDownstreamDemandNum();
        mergeCost=GraphAlgorithm.getMergeCost(this,start);
        systematicDetourCost=0.0;

        dmr.setCurrDetourCost(this,dmr.getMyNode(this),mergeCost);
        dmr.setPreNode(this,start,dummyNode);
        dmr.setPreEdge(this,start,dummyEdge);

        detouristFinish=false;
    }

    public int getID(){
        return id;
    }
```

Detourist.java

```
public Graph getUsableGraph(){
    return usableGraph;
}

public Graph getDownstream(){
    return downstream;
}

public Graph getUpstream(){
    return upstream;
}

public DetourManager getDetourManager(){
    return dmr;
}

public Center getCenter(){
    return center;
}

public Node getMergeNode(){
    return start;
}

public Node getAccessNode(){
    return access;
}

public Edge getRuinedEdge(){
    return currRuinedEdge;
}

public double getMergeCost(){
    return mergeCost;
}

public double getSystematicDetourCost(){
    return systematicDetourCost;
}

public Graph getDetourPath(){
```

Detourist.java

```
            return detourPath;
        }

        public void run(){

            dmr.takeKey(this);

            //::map init::
            Vector edgeSet=center.getGraph().getEdgeSet();
            Edge tempEdge;
            for(int i=0;i<edgeSet.size();i++){
                tempEdge=(Edge)edgeSet.elementAt(i);
                if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
                    tempEdge.setNeutralEdge();
            }
            //::map init::

            detouring:
            while(!detouristFinish){
                if(dmr.getCurrNode(this)==supply){
                    double
sdc=dmr.getCurrDetourCost(this,dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;
                    dmr.setCurrDetourCost(this,dmr.getCurrNode(this),sdc);

                    dmr.addNode(this,supply);
                    dmr.addEdge(this,dmr.getMyEdge(this));

                    dmr.setPreNode(this,supply,dmr.getMyNode(this));
                    dmr.setPreEdge(this,supply,dmr.getMyEdge(this));

                    dmr.updatMinSDC(this,dmr.getCurrDetourCost(this,dmr.getSupply()));

                    dmr.getMyEdge(this).setDetourTestEdge(true);

                    break detouring;
                }else if(dmr.getMinSDC(this)!=0.0 &&
dmr.getMinSDC(this)<dmr.getCurrDetourCost(this,dmr.getMyNode(this))){
                    //some detourist has already found a shorter detour path
                    break detouring;
                }else{
                    dmr.addNode(this,dmr.getCurrNode(this));
                    dmr.addEdge(this,dmr.getMyEdge(this));
```

Detourist.java

```
                dmr.setPreNode(this,dmr.getCurrNode(this),dmr.getMyNode(this));
                dmr.setPreEdge(this,dmr.getCurrNode(this),dmr.getMyEdge(this));

                double
sdc=dmr.getCurrDetourCost(this,dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;
                dmr.setCurrDetourCost(this,dmr.getCurrNode(this),sdc);

                dmr.getMyEdge(this).setDetourTestEdge(true);
                dmr.setMyNode(this,dmr.getCurrNode(this));

                //find currEdge & currNode, assign new myNode
                dijkstra(null);

                try{
                    sleep(1);
                }catch(InterruptedException ex){
                }

            }

        }

        //set access node
        Vector detourPathNodeSet=new Vector(),detourPathEdgeSet=new Vector();
        Node tempCurrNode=dmr.getSupply();
        detourPathNodeSet.addElement(tempCurrNode);

        Edge tempPreEdge;
        Node n1,n2;
        do{
            tempPreEdge=dmr.getPreEdge(this,tempCurrNode);

            if(!detourPathEdgeSet.contains(tempPreEdge))
                detourPathEdgeSet.addElement(tempPreEdge);

            n1=tempPreEdge.getN1();
            n2=tempPreEdge.getN2();

            if(!detourPathNodeSet.contains(n1))
                detourPathNodeSet.addElement(n1);
            if(!detourPathNodeSet.contains(n2))
```

Detourist.java

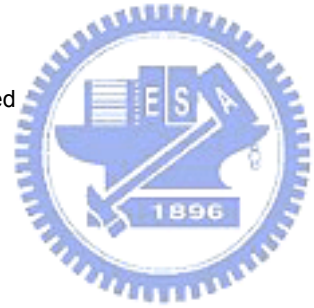
```
                detourPathNodeSet.addElement(n2);

            if(upstreamNodeSet.contains(n1) && !upstreamNodeSet.contains(n2)){
                access=n1;
            }else if(upstreamNodeSet.contains(n2) && !upstreamNodeSet.contains(n1)){
                access=n2;
            }else{
                access=null;
            }

            tempCurrNode=dmr.getPreNode(this,tempCurrNode);
        }while(tempCurrNode!=start);

        if(!detourPathNodeSet.contains(start))
            detourPathNodeSet.addElement(start);

        //set detourPath
        detourPath=new Graph(detourPathNodeSet,detourPathEdgeSet);

        //set sdc
        systematicDetourCost=dmr.getCurrDetourCost(this,supply);
        dmr.setSystematicDetourCost(this,systematicDetourCost);

        dmr.updateCurrRuinedEdgeFinish();
        dmr.updateDetourCenter(this);

        dmr.putKey(this);
    }

    void dijkstra(Edge canceledEdge){

        //find out the best incident edge
        //define new myEdge & currNode
        Vector tempRouteNodeSet=(Vector)dmr.getRouteNodeSet(this);

        Vector incidentEdgeSet=GraphAlgorithm.getIncidentEdges(this,tempRouteNodeSet);
        Vector exclusiveIncidentEdgeSet=new Vector();
        for(int i=0;i<incidentEdgeSet.size();i++){
            if(!dmr.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
                exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
            }
        }
```

Detourist.java

```
        Edge tempEdge;
        Node n1,n2;

        if(exclusiveIncidentEdgeSet.size()>1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            dmr.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                dmr.setMyNode(this,n1);
                dmr.setCurrNode(this,n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                dmr.setMyNode(this,n2);
                dmr.setCurrNode(this,n1);
            }else{
                //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }

            double
min=dmr.getCurrDetourCost(this,dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;

            for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    if((dmr.getCurrDetourCost(this,n1)+tempEdge.getWeight()*downstreamDemandNum)<min){
                        min=dmr.getCurrDetourCost(this,n1)+tempEdge.getWeight()*downstreamDemandNum;
                        dmr.setMyEdge(this,tempEdge);
                        dmr.setMyNode(this,n1);
                        dmr.setCurrNode(this,n2);
                    }
                }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                    if((dmr.getCurrDetourCost(this,n2)+tempEdge.getWeight()*downstreamDemandNum)<min){
                        min=dmr.getCurrDetourCost(this,n2)+tempEdge.getWeight()*downstreamDemandNum;
                        dmr.setMyEdge(this,tempEdge);
                        dmr.setMyNode(this,n2);
                        dmr.setCurrNode(this,n1);
                    }
                }else{
                    //dijkstra error 2: incident edge error!
```

Detourist.java

```
                    }
                }

            }else if(exclusiveIncidentEdgeSet.size()==1){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
                dmr.setMyEdge(this,tempEdge);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    dmr.setMyNode(this,n1);
                    dmr.setCurrNode(this,n2);
                }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                    dmr.setMyNode(this,n2);
                    dmr.setCurrNode(this,n1);
                }else{
                    //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
                }
            }else{
                detouristFinish=true;
                //renders all nodes visited
            }
        }
}
```



DetourManager.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Graph;
import java.util.Vector;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class DetourManager extends Thread{

    int id;
    Center center;
    Node supply;
    Graph graph,fastTree,territory;
    Vector edgeSet,nodeSet,fastTreeEdgeSet,fastTreeNodeSet,detourists,territoryNodeSet,territoryEdgeSet;
    int detouristNum,downstreamDemandNum;

    //nodeDept
    //nodeDept[detourist][node][0]: currDetourDist
    //nodeDept[detourist][node][1]: preNode
    //nodeDept[detourist][node][2]: preEdge
    Object[][][] nodeDept;

    //detourDept
    //detourDept[detourist][0]: myNode(Node)
    //detourDept[detourist][1]: myEdge(Edge)
    //detourDept[detourist][2]: currNode(Node)
    //detourDept[detourist][3]: routeSet(Graph)
    //detourDept[detourist][4]: detourLength(Double)
    //deoutrDept[detourist][5]: bridge(Graph)
    Object[][] detourDept;

    //sdcDept
    //sdcDept[ruinedEdge][0]: minSDC
    Object[][] sdcDept;

    Edge currRuinedEdge;
    Graph currUpstream,currDownstream;
    Vector currDownstreamNodeSet;

    boolean currRuinedEdgeFinish,available;
```

DetourManager.java

```
    //one manager controls one territory
    public DetourManager(int id,Center center,Node supply){
        this.id=id;
        this.center=center;
        this.supply=supply;

        graph=center.getGraph();
        edgeSet=graph.getEdgeSet();
        nodeSet=graph.getNodeSet();

        fastTree=center.getFastTree(supply);
        fastTreeEdgeSet=fastTree.getEdgeSet();
        fastTreeNodeSet=fastTree.getNodeSet();

//          detouristNum=fastTreeNodeSet.size();

        nodeDept=new Object[fastTreeNodeSet.size()][nodeSet.size()][3];
        detourDept=new Object[fastTreeNodeSet.size()][6];

        if(fastTreeEdgeSet.size()!=0){
            Edge tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(0);
            int maxEdgeLabel=tempFastTreeEdge.getLabel();
            for(int j=1;j<fastTreeEdgeSet.size();j++){
                tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(j);
                if(tempFastTreeEdge.getLabel()>maxEdgeLabel)
                    maxEdgeLabel=tempFastTreeEdge.getLabel();
            }
            sdcDept=new Object[maxEdgeLabel+1][1];
        }
    }

    public void run(){

        if(fastTreeEdgeSet==null){
            center.updateDetourCondition();
            destroy();
        }
        center.takeKey(this);

        for(int i=0;i<fastTreeEdgeSet.size();i++){
```

DetourManager.java

```
//initialization
currRuinedEdgeFinish=false;
available=true;

for(int j=0;j<fastTreeNodeSet.size();j++){
    Node tempNode;
    for(int k=0;j<fastTreeNodeSet.size();j++){
        tempNode=(Node)fastTreeNodeSet.elementAt(j);
        nodeDept[j][tempNode.getLabel()][0]=new Double(0.0);
    }
}

for(int j=0;j<detouristNum;j++){
    detourDept[j][0]=null;
    detourDept[j][1]=null;
    detourDept[j][2]=null;
    detourDept[j][3]=null;
    detourDept[j][4]=new Double(0.0);
    detourDept[j][5]=null;
}

Edge tempFastTreeEdge;
for(int j=0;j<fastTreeEdgeSet.size();j++){
    tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(j);
    sdcDept[tempFastTreeEdge.getLabel()][0]=new Double(0.0);
}

currRuinedEdge=(Edge)fastTreeEdgeSet.elementAt(i);
currUpstream=GraphAlgorithm.getSubtreeWithSupply(fastTree,currRuinedEdge);
currDownstream=GraphAlgorithm.getSubtreeWithoutSupply(fastTree,currRuinedEdge);
currDownstreamNodeSet=currDownstream.getNodeSet();
detouristNum=currDownstreamNodeSet.size();

Node tempNode;
downstreamDemandNum=0;
for(int k=0;k<currDownstreamNodeSet.size();k++){
    tempNode=(Node)currDownstreamNodeSet.elementAt(k);
    if(tempNode.isDemand())
        downstreamDemandNum++;
}

//one detourist tests from one node when one currRuinedEdge is simulated
```

DetourManager.java

```
        Detourist tempDetourist;
        detourists=new Vector();
        for(int j=0;j<currDownstreamNodeSet.size();j++){
            tempNode=(Node)currDownstreamNodeSet.elementAt(j);
            tempDetourist=new Detourist(j,tempNode,this);
            detourists.addElement(tempDetourist);
            tempDetourist.start();
        }

        //dmr waiting the currRuinedEdge finish
        while(!currRuinedEdgeFinish){
        }

        //currRuinedEdge finish: all situations simulated
        //set detour edges on the shortest detour route
        Detourist bestDetourist=(Detourist)detourists.elementAt(0);
        Detourist tempDetourist1;
        double minSDC=bestDetourist.getSystematicDetourCost();
        for(int j=1;j<detourists.size();j++){
            tempDetourist1=(Detourist)detourists.elementAt(j);
            if(tempDetourist1.getSystematicDetourCost()<minSDC){
                minSDC=tempDetourist1.getSystematicDetourCost();
                bestDetourist=tempDetourist1;
            }
        }

        Vector bestDetourPathEdgeSet=bestDetourist.getDetourPath().getEdgeSet();
        Edge tempEdge1;
        for(int j=0;j<bestDetourPathEdgeSet.size();j++){
            tempEdge1=(Edge)bestDetourPathEdgeSet.elementAt(j);
            tempEdge1.setDetourEdge();
        }
    }

    //::maCenter[supply][1]: territory(Graph) 2ECON::
    Graph territory=fastTree;
    Edge tempRuinedEdge;
    Graph tempDetourPath;
    Vector tempDetourPathNodeSet,tempDetourPathEdgeSet;
    Node tempDetourPathNode;
    Edge tempDetourPathEdge;
    for(int i=0;i<fastTreeEdgeSet.size();i++){
```

DetourManager.java

```
                tempRuinedEdge=(Edge)fastTreeEdgeSet.elementAt(i);
                tempDetourPath=center.getDetourPath(tempRuinedEdge);
                if(tempDetourPath!=null){
                    tempDetourPathNodeSet=tempDetourPath.getNodeSet();
                    tempDetourPathEdgeSet=tempDetourPath.getEdgeSet();
                    if(tempDetourPathNodeSet!=null){
                        for(int j=0;j<tempDetourPathNodeSet.size();j++){
                            tempDetourPathNode=(Node)tempDetourPathNodeSet.elementAt(j);
                            if(!territory.hasNode(tempDetourPathNode))
                                territory.addNode(tempDetourPathNode);
                        }
                    }
                    if(tempDetourPathEdgeSet!=null){
                        for(int j=0;j<tempDetourPathEdgeSet.size();j++){
                            tempDetourPathEdge=(Edge)tempDetourPathEdgeSet.elementAt(j);
                            if(!territory.hasEdge(tempDetourPathEdge))
                                territory.addEdge(tempDetourPathEdge);
                        }
                    }
                }else{
                    //tempDetourPath is null!
                }
            }

            center.setTerritory(supply,territory);
            //::maCenter[supply][1]: territory(Graph) 2ECON::

            center.updateDetourCondition();
            center.putKey(this);
        }

        //detourDept method:
        public synchronized void takeKey(Detourist detourist){
            if(!currRuinedEdgeFinish){
                while(!available){
                    try{
                        wait(1);
                    }catch(InterruptedException e){
                        //takeKey: cannot wait!
                    }
                }
                available=false;
```

DetourManager.java

```
        }else{
            available=false;
        }
    }

    public synchronized void putKey(Detourist detourist){
        if(!isCurrRuinedEdgeFinish()){
            while(available){
                try{
                    wait(1);
                }catch(InterruptedException e){
                    //putKey: cannot wait!
                }
            }
            available=true;
        }else{
            available=true;
        }
    }

    public synchronized boolean isFinished(){
        return currRuinedEdgeFinish;
    }

    void sendDetourist(Graph downstream){
        Vector fastTreeNodeSet=downstream.getNodeSet();
        Node tempNode;
        for(int i=0;i<fastTreeNodeSet.size();i++){
            tempNode=(Node)fastTreeNodeSet.elementAt(i);
            new Detourist(i,tempNode,this).start();
        }
    }

    public synchronized void setCurrRuinedEdgeFinish(boolean currRuinedEdgeFinish){
        this.currRuinedEdgeFinish=currRuinedEdgeFinish;
    }

    public Edge getCurrRuinedEdge(){
        return currRuinedEdge;
    }

    public Graph getCurrUpstream(){
```

DetourManager.java

```
        return currUpstream;
    }

    public Graph getCurrDownstream(){
        return currDownstream;
    }

    public Graph getGraph(){
        return center.getGraph();
    }

    public void setPreNode(Detourist detourist,Node node,Node preNode){
        nodeDept[detourist.getID()][node.getLabel()][1]=preNode;
    }

    public Node getPreNode(Detourist detourist,Node node){
        return (Node)nodeDept[detourist.getID()][node.getLabel()][1];
    }

    public void setPreEdge(Detourist detourist,Node node,Edge preEdge){
        nodeDept[detourist.getID()][node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Detourist detourist,Node node){
        return (Edge)nodeDept[detourist.getID()][node.getLabel()][2];
    }

    public int getDownstreamDemandNum(){
        return downstreamDemandNum;
    }

    public Center getCenter(){
        return center;
    }

    public Node getSupply(){
        return supply;
    }

    public int getID(){
        return id;
    }
```

DetourManager.java

```
public synchronized void updateCurrRuinedEdgeFinish(){
    detouristNum--;
    if(detouristNum==0){
        currRuinedEdgeFinish=true;
    }else{
        currRuinedEdgeFinish=false;
    }
}

public synchronized boolean isCurrRuinedEdgeFinish(){
    return currRuinedEdgeFinish;
}

public synchronized void updateDetourCenter(Detourist detourist){
    Center center=detourist.getCenter();
    DetourManager dmr=detourist.getDetourManager();
    Edge ruinedEdge=detourist.getRuinedEdge();

    //detourCenter[edge][0]: downstream(Graph)
    //detourCenter[edge][1]: upstream(Graph)
    //detourCenter[edge][2]: mergeNode(Node)
    //detourCenter[edge][3]: accessNode(Node)
    //detourCenter[edge][4]: detourPath(Graph)
    //detourCenter[edge][5]: systematicDetourCost(Double)
    //detourCenter[edge][6]: mergeCost(Vector)

    //maCenter[supply][1]: territory(Graph) 2ECON

    double detouristDetourLength=dmr.getSystematicDetourCost(detourist);
    double centerDetourLengthRecord=center.getSystematicDetourCost(detourist.getRuinedEdge());
    if(centerDetourLengthRecord==0.0 || detouristDetourLength<centerDetourLengthRecord){

        center.setDownstream(ruinedEdge,detourist.getDownstream());
        center.setUpstream(ruinedEdge,detourist.getUpstream());
        center.setMergeNode(ruinedEdge,detourist.getMergeNode());
        center.setAccessNode(ruinedEdge,detourist.getAccessNode());
        center.setDetourPath(ruinedEdge,detourist.getDetourPath());
        center.setSystematicDetourCost(ruinedEdge,detouristDetourLength);
        center.setMergeCost(ruinedEdge,detourist.getMergeCost());

        //detouristDetourLength is smaller than center record
```

DetourManager.java

```
        }else{
            //detouristDetourLength is larger than center record
        }
    }

    //detourDept method:
    //detourDept[detourist][0]: myNode(Node)
    public synchronized void setMyNode(Detourist detourist,Node myNode){
        detourDept[detourist.getID()][0]=myNode;
    }

    public synchronized Node getMyNode(Detourist detourist){
        Node myNode=(Node)detourDept[detourist.getID()][0];
        return myNode;
    }

    //detourDept[detourist][1]: myEdge(Edge)
    public synchronized void setMyEdge(Detourist detourist,Edge myEdge){
        detourDept[detourist.getID()][1]=myEdge;
    }

    public synchronized Edge getMyEdge(Detourist detourist){
        Edge myEdge=(Edge)detourDept[detourist.getID()][1];
        return myEdge;
    }

    //detourDept[detourist][2]: currNode(Node)
    public synchronized void setCurrNode(Detourist detourist,Node currNode){
        detourDept[detourist.getID()][2]=currNode;
    }

    public synchronized Node getCurrNode(Detourist detourist){
        Node currNode=(Node)detourDept[detourist.getID()][2];
        return currNode;
    }

    //detourDept[detourist][3]: routeSet(Graph)
    public synchronized void setRouteSet(Detourist detourist,Graph routeSet){
        detourDept[detourist.getID()][3]=routeSet;
    }

    public synchronized void setRouteSet(Detourist detourist,Vector routeNodeSet,Vector routeEdgeSet){
```

DetourManager.java

```
        detourDept[detourist.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
    }

    public synchronized Graph getRouteSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        return routeSet;
    }

    public synchronized Vector getRouteNodeSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        Vector routeNodeSet=routeSet.getNodeSet();
        return routeNodeSet;
    }

    public synchronized Vector getRouteEdgeSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        Vector routeEdgeSet=routeSet.getEdgeSet();
        return routeEdgeSet;
    }

    public synchronized void addEdge(Detourist detourist,Edge edge){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Detourist detourist,Node node){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeEdge(Detourist detourist,Edge edge){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
```

DetourManager.java

```
        if(routeEdgeSet.contains(edge)){
            routeEdgeSet.removeElement(edge);
        }
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeSubtree(Detourist detourist,Graph subtree){
        Graph routeSet=(Graph)this.detourDept[detourist.getID()][3];
        Vector tempNodeSet1=routeSet.getNodeSet();
        Vector tempEdgeSet1=routeSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            boolean nodeExist=tempNodeSet1.removeElement(tempNode);
        }

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            boolean edgeExist=tempEdgeSet1.removeElement(tempEdge);
        }
    }

    public synchronized void addSubtree(Detourist detourist,Graph subtree){

        Graph visitedRouteSet=(Graph)this.detourDept[detourist.getID()][3];
        Vector tempNodeSet1=visitedRouteSet.getNodeSet();
        Vector tempEdgeSet1=visitedRouteSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            if(tempNodeSet1.contains(tempNode)){
            }else{
                tempNodeSet1.addElement(tempNode);
            }
        }
```

DetourManager.java

```
        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            if(tempEdgeSet1.contains(tempEdge)){
            }else{
                tempEdgeSet1.addElement(tempEdge);
            }
        }
    }

    //detourDept[detourist][4]: detourLength(Double)
    public void setSystematicDetourCost(Detourist detourist, double sdc){
        detourDept[detourist.getID()][4]=new Double(sdc);
    }

    public double getSystematicDetourCost(Detourist detourist){
        Double sdc=(Double)detourDept[detourist.getID()][4];
        return sdc.doubleValue();
    }


    public void setCurrDetourCost(Detourist detourist,Node node,double sdc){
        nodeDept[detourist.getID()][node.getLabel()][0]=new Double(sdc);
    }

    public double getCurrDetourCost(Detourist detourist,Node node){
        Double tempDouble=(Double)nodeDept[detourist.getID()][node.getLabel()][0];
        return tempDouble.doubleValue();
    }

    //detourCenter:
    public synchronized void updateMinSDC(double sdc){
        double currSDC=center.getSystematicDetourCost(currRuinedEdge);
        if(currSDC==0.0 || sdc<currSDC)
            center.setSystematicDetourCost(currRuinedEdge,sdc);
    }

    public synchronized double getMinSDC(){
        return center.getSystematicDetourCost(currRuinedEdge);
    }
```

DetourManager.java

```
	//deoutrDept[detourist][5]: bridge(Graph)
	//sdcDept
	//sdcDept[ruinedEdge][0]: minSDC
	public synchronized void updatMinSDC(Detourist detourist,double sdc){
		Edge ruinedEdge=detourist.getRuinedEdge();
		DetourManager dmr=detourist.getDetourManager();
		double currSDC=dmr.getMinSDC(detourist);
		if(sdc<currSDC)
			sdcDept[ruinedEdge.getLabel()][0]=new Double(sdc);
	}

	public synchronized double getMinSDC(Detourist detourist){
		Edge ruinedEdge=detourist.getRuinedEdge();
		Double minSDC=(Double)sdcDept[ruinedEdge.getLabel()][0];
		return minSDC.doubleValue();
	}
}
```

Helper.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Node;
import emnet.graph.Graph;
import emnet.graph.Edge;

public class Helper extends Thread{
	int id;
	MAManager maMr;
	Node start,source;

	Node currNode,preNode,myNode;
	Edge preEdge,myEdge;

	Graph maPath;

	//maCost = mergeCost + demandNum * bridgeLength(maPath)
	double maCost;

	//for merge cost
	//walkerCenter[node][]:
	//nodeDept
	//nodeDept[node][0]: currCost
	//nodeDept[node][1]: preNode
	//nodeDept[node][2]: preEdge
	Object[][] nodeDept;

	//walkerDept
	//walkerDept[0]: myNode(Node)
	//walkerDept[1]: myEdge(Edge)
	//walkerDept[2]: currNode(Node)
	//walkerDept[3]: routeSet(Graph)
	Object[] walkerDept;

	boolean helperFinish,longer,exclusiveIsZero;

	public Helper(int id,Node start,MAManager maMr){

		this.id=id;
		this.start=start;
```

```
        this.maMr=maMr;
        this.source=null;

        //nodeDept
        nodeDept=new Object[maMr.getGraph().getNodeSet().size()][3];

        //walkerDept
        walkerDept=new Object[4];

        Node dummyNode=maMr.getCenter().getDummyNode();
        dummyNode.setDummy();
        int edgeNum=maMr.getGraph().getEdgeSet().size();
        Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        maMr.getGraph().addNode(dummyNode);
        maMr.getGraph().addEdge(dummyEdge);

        //helper init, set to maMr
        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);

        maMr.setRouteSet(this,routeNodeSet,routeEdgeSet);
        maMr.setMyNode(this,dummyNode);
        maMr.setMyEdge(this,dummyEdge);
        maMr.setCurrNode(this,start);

        double mergeCost=getECONMergeCost(start);

        maMr.setCurrMACost(this,maMr.getMyNode(this),mergeCost);
        maMr.setPreNode(this,start,dummyNode);
        maMr.setPreEdge(this,start,dummyEdge);
        maCost=0.0;
        helperFinish=false;
        longer=false;
        exclusiveIsZero=false;
    }

    public int getID(){
        return id;
```

```
}

public Node getStart(){
    return start;
}

public MAManager getMAManger(){
    return maMr;
}

public double getMACost(){
    return maCost;
}

public Graph getMAPath(){
    return maPath;
}

public Node getSource(){
    return source;
}

public void run(){
    maMr.takeKey(this);

    //::map init::
    Vector edgeSet=maMr.getCenter().getGraph().getEdgeSet();
    Edge tempEdge;
    for(int i=0;i<edgeSet.size();i++){
        tempEdge=(Edge)edgeSet.elementAt(i);
        if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
            tempEdge.setNeutralEdge();
    }
    //::map init::

    maHelping:
    while(!helperFinish){

        if(maMr.getCurrNode(this).isSupply() && maMr.getCurrNode(this)!=maMr.getSupply()){

            source=maMr.getCurrNode(this);
```

Helper.java

```
                double
currMACost=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemanNum();
                maMr.setCurrMACost(this,maMr.getCurrNode(this),currMACost);

                maMr.addNode(this,maMr.getCurrNode(this));
                maMr.addEdge(this,maMr.getMyEdge(this));

                maMr.setPreNode(this,maMr.getCurrNode(this),maMr.getMyNode(this));
                maMr.setPreEdge(this,maMr.getCurrNode(this),maMr.getMyEdge(this));

                maMr.updateMinMACost(maMr.getCurrMACost(this,maMr.getCurrNode(this)));

                maMr.getMyEdge(this).setMATestEdge(true);

                break maHelping;
            }else if(maMr.getMinMACost()!=0.0 &&
maMr.getMinMACost()<maMr.getCurrMACost(this,maMr.getMyNode(this))){
                //some detourist has already found a shorter detour path
                longer=true;
                break maHelping;
            }else{
                maMr.addNode(this,maMr.getCurrNode(this));
                maMr.addEdge(this,maMr.getMyEdge(this));

                maMr.setPreNode(this,maMr.getCurrNode(this),maMr.getMyNode(this));
                maMr.setPreEdge(this,maMr.getCurrNode(this),maMr.getMyEdge(this));

                double
currMACost=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemanNum();
                maMr.setCurrMACost(this,maMr.getCurrNode(this),currMACost);

                maMr.getMyEdge(this).setMATestEdge(true);
                maMr.setMyNode(this,maMr.getCurrNode(this));

                //find currEdge & currNode, assign new myNode
                dijkstra();

                try{
                    sleep(1);
                }catch(InterruptedException ex){
                    System.out.println("maMr "+maMr.getID()+", helper "+getID()+" cannot sleep: "+ex);
                }
```

Helper.java

```
            }
        }

        //helper finished!
        //maPath setting:
        if(!longer && !exclusiveIsZero){
            Vector maPathNodeSet=new Vector();
            Vector maPathEdgeSet=new Vector();

            Node tempCurrNode=maMr.getCurrNode(this);
            Edge tempPreEdge;

            maPathSetting:
            do{
                if(!maPathNodeSet.contains(tempCurrNode))
                    maPathNodeSet.addElement(tempCurrNode);

                tempPreEdge=maMr.getPreEdge(this,tempCurrNode);

                //tempPreEdge!=null
                if(!tempPreEdge.isDummyEdge()){
                    if(!maPathEdgeSet.contains(tempPreEdge))
                        maPathEdgeSet.addElement(tempPreEdge);

                    tempCurrNode=maMr.getPreNode(this,tempCurrNode);
                }else{
                    break maPathSetting;
                }

            }while(tempCurrNode!=start);

            if(!maPathNodeSet.contains(start))
                maPathNodeSet.addElement(start);

            maPath=new Graph(maPathNodeSet,maPathEdgeSet);

            //update mutual assistant path to center
            //maCost setting:
            maCost=maMr.getCurrMACost(this,maMr.getCurrNode(this));
        }
        maMr.updateMAFinish();
        maMr.putKey(this);
```

Helper.java

```
    }

    void dijkstra(){
        //find out the best incident edge
        //define new myEdge & currNode
        Graph tempRouteSet=(Graph)maMr.getRouteSet(this);
        Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();

        Vector incidentEdgeSet=getIncidentEdgeSet(maMr.getUsableGraph(),tempRouteNodeSet);

        Vector exclusiveIncidentEdgeSet=new Vector();
        for(int i=0;i<incidentEdgeSet.size();i++){
            if(!maMr.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
                exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
            }
        }

        Edge tempEdge;
        Node n1,n2;

        if(exclusiveIncidentEdgeSet.size()>1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            maMr.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                maMr.setMyNode(this,n1);
                maMr.setCurrNode(this,n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                maMr.setMyNode(this,n2);
                maMr.setCurrNode(this,n1);
            }else{
                //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }
            double
min=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemanNum();
            for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    if(maMr.getCurrMACost(this,n1)+tempEdge.getWeight()*maMr.getDemanNum()<min){
```

Helper.java

```
                        min=maMr.getCurrMACost(this,n1)+tempEdge.getWeight()*maMr.getDemanNum();
                        maMr.setMyEdge(this,tempEdge);
                        maMr.setMyNode(this,n1);
                        maMr.setCurrNode(this,n2);
                    }
                }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                    if(maMr.getCurrMACost(this,n2)+tempEdge.getWeight()*maMr.getDemanNum()<min){
                        min=maMr.getCurrMACost(this,n2)+tempEdge.getWeight()*maMr.getDemanNum();
                        maMr.setMyEdge(this,tempEdge);
                        maMr.setMyNode(this,n2);
                        maMr.setCurrNode(this,n1);
                    }
                }else{
                    //dijkstra error 2: incident edge error!"
                }
            }
        }else if(exclusiveIncidentEdgeSet.size()==1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            maMr.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                maMr.setMyNode(this,n1);
                maMr.setCurrNode(this,n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                maMr.setMyNode(this,n2);
                maMr.setCurrNode(this,n1);
            }else{
                //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }
        }else{
            exclusiveIsZero=true;
            helperFinish=true;
            //renders all nodes visited
        }

    }

    double getECONMergeCost(Node node){
        Walker walker=new Walker(this,node);
        walker.start();
```

Helper.java

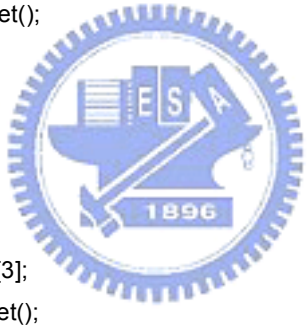
```
        while(!walker.isFinish()){
            //wait for walker to calculate merge cost
        }
        return walker.getTerritoryMergeCost();
    }

    Vector getIncidentEdgeSet(Graph usableGraph,Vector routeNodeSet){
        Edge myEdge=maMr.getMyEdge(this);

        Vector incidentEdges=new Vector();
        Edge tempEdge;
        Node n1,n2;

        for(int i=0;i<routeNodeSet.size();i++){
            Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)routeNodeSet.elementAt(i));
            for(int j=0;j<tempEdgeSet.size();j++){
                tempEdge=(Edge)tempEdgeSet.elementAt(j);
                if(!incidentEdges.contains(tempEdge))
                    incidentEdges.addElement(tempEdge);

                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(routeNodeSet.contains(n1) && routeNodeSet.contains(n2))
                    incidentEdges.removeElement(tempEdge);
                if(n1==maMr.getCenter().getDummyNode() || n2==maMr.getCenter().getDummyNode())
                    incidentEdges.removeElement(tempEdge);
            }
        }
        if(incidentEdges.contains(myEdge))
            incidentEdges.removeElement(myEdge);

        return incidentEdges;
    }

    //walker methods:
    //nodeDept[node][0]: currCost
    public void setCurrCost(Node node,double cost){
        nodeDept[node.getLabel()][0]=new Double(cost);
    }

    public double getCurrCost(Node node){
        Double tempDouble=(Double)nodeDept[node.getLabel()][0];
```

Helper.java

```
        return tempDouble.doubleValue();
    }

    //nodeDept[node][1]: preNode
    public void setPreNode(Node node,Node preNode){
        nodeDept[node.getLabel()][1]=preNode;
    }

    public Node getPreNode(Node node){
        return (Node)nodeDept[node.getLabel()][1];
    }

    //nodeDept[node][2]: preEdge
    public void setPreEdge(Node node,Edge preEdge){
        nodeDept[node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Node node){
        return (Edge)nodeDept[node.getLabel()][2];
    }

    //walkerDept[0]: myNode(Node)
    public void setMyNode(Node node){
        walkerDept[0]=node;
    }

    public Node getMyNode(){
        return (Node)walkerDept[0];
    }

    //walkerDept[1]: myEdge(Edge)
    public void setMyEdge(Edge edge){
        walkerDept[1]=edge;
    }

    public Edge getMyEdge(){
        return (Edge)walkerDept[1];
    }

    //walkerDept[2]: currNode(Node)
    public void setCurrNode(Node node){
        walkerDept[2]=node;
```

Helper.java

```
    }

    public Node getCurrNode(){
        return (Node)walkerDept[2];
    }

    //walkerDept[3]: routeSet(Graph)
    public void setRouteSet(Vector nodeSet,Vector edgeSet){
        walkerDept[3]=new Graph(nodeSet,edgeSet);
    }

    public Graph getRouteSet(){
        return (Graph)walkerDept[3];
    }

    public void addNode(Node node){
        Graph routeSet=(Graph)walkerDept[3];
        Vector nodeSet=routeSet.getNodeSet();
        if(!nodeSet.contains(node))
            nodeSet.addElement(node);
    }

    public void addEdge(Edge edge){
        Graph routeSet=(Graph)walkerDept[3];
        Vector edgeSet=routeSet.getEdgeSet();
        if(!edgeSet.contains(edge))
            edgeSet.addElement(edge);
    }

}
```



MAManager.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Node;
import emnet.graph.Graph;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class MAManager extends Thread{

    int id;
    Center center;

    Node supply;
    Graph territory,usableGraph;
    Vector icp,helpers,usableGraphNodeSet,usableGraphEdgeSet,interfaceNodes;
    int demandNum,helperNum;

    //nodeDept
    //nodeDept[helper][node][0]: currMACost
    //nodeDept[helper][node][1]: preNode
    //nodeDept[helper][node][2]: preEdge
    Object[][][] nodeDept;

    //maDept
    //maDept[helper][0]: myNode(Node)
    //maDept[helper][1]: myEdge(Edge)
    //maDept[helper][2]: currNode(Node)
    //maDept[helper][3]: routeSet(Graph)
    Object[][] maDept;

    //maCost
    double minMACost;

    boolean available,maFinish;

    public MAManager(int id,Center center,Node supply){
        this.id=id;
        this.center=center;
        this.supply=supply;
```

```
territory=center.getTerritory(supply);


if(territory.getDemandNodeSet().size()!=0){
    demandNum=territory.getDemandNodeSet().size();

    Vector territoryNodeSet=territory.getNodeSet();

    //icp init, usableGraph init
    Graph graph=center.getGraph();
    Vector graphEdgeSet=graph.getEdgeSet();

    usableGraphNodeSet=new Vector();
    usableGraphEdgeSet=new Vector();
    icp=new Vector();

    Edge tempEdge;
    Node n1,n2;
    for(int i=0;i<graphEdgeSet.size();i++){
        tempEdge=(Edge)graphEdgeSet.elementAt(i);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();

        if(!territoryNodeSet.contains(n1) && !territoryNodeSet.contains(n2)){
            if(!usableGraphEdgeSet.contains(tempEdge))
                usableGraphEdgeSet.addElement(tempEdge);
            if(!usableGraphNodeSet.contains(n1))
                usableGraphNodeSet.addElement(n1);
            if(!usableGraphNodeSet.contains(n2))
                usableGraphNodeSet.addElement(n2);
        }

        if(territoryNodeSet.contains(n1) && !territoryNodeSet.contains(n2)){
            if(!usableGraphEdgeSet.contains(tempEdge))
                usableGraphEdgeSet.addElement(tempEdge);
            if(!usableGraphNodeSet.contains(n1))
                usableGraphNodeSet.addElement(n1);
            if(!usableGraphNodeSet.contains(n2))
                usableGraphNodeSet.addElement(n2);
            if(!tempEdge.isDummyEdge()){
                if(!icp.contains(n1))
                    icp.addElement(n1);
```

```
                }
            }

            if(!territoryNodeSet.contains(n1) && territoryNodeSet.contains(n2)){
                if(!usableGraphEdgeSet.contains(tempEdge))
                    usableGraphEdgeSet.addElement(tempEdge);
                if(!usableGraphNodeSet.contains(n1))
                    usableGraphNodeSet.addElement(n1);
                if(!usableGraphNodeSet.contains(n2))
                    usableGraphNodeSet.addElement(n2);
                if(!tempEdge.isDummyEdge()){
                    if(!icp.contains(n2))
                        icp.addElement(n2);
                }
            }

            //while territory is not convex
            if(!territory.hasEdge(tempEdge) && !usableGraphEdgeSet.contains(tempEdge)){
                usableGraphEdgeSet.addElement(tempEdge);
                if(!usableGraphNodeSet.contains(n1))
                    usableGraphNodeSet.addElement(n1);
                if(!usableGraphNodeSet.contains(n2))
                    usableGraphNodeSet.addElement(n2);
                if(!tempEdge.isDummyEdge()){
                    if(!icp.contains(n1))
                        icp.addElement(n1);
                    if(!icp.contains(n2))
                        icp.addElement(n2);
                }
            }
        }
        usableGraph=new Graph(usableGraphNodeSet,usableGraphEdgeSet);
        interfaceNodes=GraphAlgorithm.getInterfaceNodes(territory,graph);
        helperNum=icp.size();
        nodeDept=new Object[icp.size()][graph.getNodeSet().size()][3];
        for(int i=0;i<icp.size();i++){
            for(int j=0;j<graph.getNodeSet().size();j++){
                nodeDept[i][j][0]=new Double(0.0);
            }
        }
        maDept=new Object[icp.size()][4];
        minMACost=0.0;
```

MAManager.java

```
                available=true;
                maFinish=false;
            }else{
                maFinish=true;
            }

        }

        public void run(){
            center.takeKey(this);

            //map init
            Vector edgeSet=center.getGraph().getEdgeSet();
            Edge tempEdge;
            for(int i=0;i<edgeSet.size();i++){
                tempEdge=(Edge)edgeSet.elementAt(i);
                if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
                    tempEdge.setNeutralEdge();
            }

            if(territory.getDemandNodeSet().size()!=0){
                helpers=new Vector();
                Node tempNode;
                Helper tempHelper;
                for(int i=0;i<icp.size();i++){
                    tempNode=(Node)icp.elementAt(i);
                    tempHelper=new Helper(i,tempNode,this);
                    helpers.addElement(tempHelper);
                    tempHelper.start();
                }

                while(!maFinish){
                    //watching
                }

                //finding supply source finished: all situations simulated
                //set ma edges on the shortest ma route
                //find a subject helper
                Helper bestHelper=(Helper)helpers.elementAt(0);
                for(int j=1;j<helpers.size();j++){
                    if(bestHelper.getMACost()==0.0)
                        bestHelper=(Helper)helpers.elementAt(j);
```

```
        }

        //find a competitor helper
        Helper tempHelper1;
        double minMACost=bestHelper.getMACost();
        for(int j=1;j<helpers.size();j++){
            tempHelper1=(Helper)helpers.elementAt(j);
            if(tempHelper1.getMACost()!=0.0 && tempHelper1.getMACost()<minMACost){
                minMACost=tempHelper1.getMACost();
                bestHelper=tempHelper1;
            }
        }

        updateMACenter(bestHelper);

        Vector bestMAPathEdgeSet=bestHelper.getMAPath().getEdgeSet();
        if(bestMAPathEdgeSet.size()>0){
            Edge tempEdge1;
            Node n1,n2;
            Node source=bestHelper.getSource();
            Graph sourceTerritory=center.getTerritory(source);
            for(int j=0;j<bestMAPathEdgeSet.size();j++){
                tempEdge1=(Edge)bestMAPathEdgeSet.elementAt(j);
                tempEdge1.setMAEdge();

                n1=tempEdge1.getN1();
                n2=tempEdge1.getN2();
                if(territory.hasNode(n1) && !territory.hasNode(n2))
                    n1.setMerge();
                if(territory.hasNode(n2) && !territory.hasNode(n1))
                    n2.setMerge();
                if(sourceTerritory.hasNode(n1) && !sourceTerritory.hasNode(n2))
                    n1.setAccess();
                if(sourceTerritory.hasNode(n2) && !sourceTerritory.hasNode(n1))
                    n2.setAccess();

                if(j==(bestMAPathEdgeSet.size()-1)){
                    if(territory.hasNode(n1) && sourceTerritory.hasNode(n1)){
                        n1.setAccess();
                        n1.setMerge();
                    }
                    if(territory.hasNode(n2) && sourceTerritory.hasNode(n2)){
```

MAManager.java

```
                                n2.setAccess();
                                n2.setMerge();
                            }
                        }
                    }
                }else{
                    bestHelper.getSource().setSource();
                }
            }else{
                //demandNum=0

                //mutualAssistantCenter
                //maCenter[supply][2]: source(Node)
                //maCenter[supply][3]: icpSet(Vector)
                //maCenter[supply][4]: maPath(Graph)
                //maCenter[supply][7]: maCost > source to supply
                center.setSource(supply,supply);
                center.setICPSet(supply,territory.getNodeSet());
                center.setMAPath(supply,territory);
                center.setMACost(supply,0.0);
            }

            //ma manager finish its job!
            center.updateMACondition();
            center.putKey(this);
        }

        public int getID(){
            return id;
        }

        public Center getCenter(){
            return center;
        }

        public Node getSupply(){
            return supply;
        }

        public Graph getGraph(){
            return center.getGraph();
        }
```

MAManager.java

```
public Graph getUsableGraph(){
	return usableGraph;
}

public Graph getTerritory(){
	return territory;
}

public int getDemanNum(){
	return demandNum;
}

public int getICP(){
	return interfaceNodes.size();
}

public synchronized void updateMinMACost(double currMACost){
	if(minMACost==0.0 || currMACost<minMACost)
		minMACost=currMACost;
}

public double getMinMACost(){
	return minMACost;
}

public void updateMAFinish(){
	helperNum--;
	if(helperNum==0){
		maFinish=true;
	}
}

public synchronized void takeKey(Helper helper){
	if(!maFinish){
		while(!available){
			try{
				wait(1);
			}catch(InterruptedException e){
				//takeKey: cannot wait!
			}
		}
```

MAManager.java

```
            available=false;
        }else{
            available=false;
            //helper took the key, but center is finished!
        }
    }

    public synchronized void putKey(Helper helper){
        if(!maFinish){
            while(available){
                try{
                    //helper is waiting to put...
                    wait(1);
                }catch(InterruptedException e){
                    //putKey: cannot wait!
                }
            }
            //helper put the key!
            available=true;
        }else{
            //helper put the key, maFinish=true!
            available=true;
        }
    }

    //nodeDept
    //nodeDept[helper][node][0]: currMACost
    public void setCurrMACost(Helper helper,Node node,double maCost){
        nodeDept[helper.getID()][node.getLabel()][0]=new Double(maCost);
    }

    public double getCurrMACost(Helper helper,Node node){
        Double maCost=(Double)nodeDept[helper.getID()][node.getLabel()][0];
        return maCost.doubleValue();
    }

    //nodeDept[helper][node][1]: preNode
    public void setPreNode(Helper helper,Node node,Node preNode){
        nodeDept[helper.getID()][node.getLabel()][1]=preNode;
    }

    public Node getPreNode(Helper helper,Node node){
```

MAManager.java

```
        return (Node)nodeDept[helper.getID()][node.getLabel()][1];
    }

    //nodeDept[helper][node][2]: preEdge
    public void setPreEdge(Helper helper,Node node,Edge preEdge){
        nodeDept[helper.getID()][node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Helper helper,Node node){
        return (Edge)nodeDept[helper.getID()][node.getLabel()][2];
    }

    //maDept
    //maDept[helper][0]: myNode(Node)
    public void setMyNode(Helper helper,Node myNode){
        maDept[helper.getID()][0]=myNode;
    }

    public Node getMyNode(Helper helper){
        return (Node)maDept[helper.getID()][0];
    }

    //maDept[helper][1]: myEdge(Edge)
    public void setMyEdge(Helper helper,Edge myEdge){
        maDept[helper.getID()][1]=myEdge;
    }

    public Edge getMyEdge(Helper helper){
        return (Edge)maDept[helper.getID()][1];
    }

    //maDept[helper][2]: currNode(Node)
    public void setCurrNode(Helper helper,Node currNode){
        maDept[helper.getID()][2]=currNode;
    }

    public Node getCurrNode(Helper helper){
        return (Node)maDept[helper.getID()][2];
    }

    //maDept[helper][3]: routeSet(Graph)
    public void setRouteSet(Helper helper,Vector routeNodeSet,Vector routeEdgeSet){
```

```
        maDept[helper.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
    }

    public Graph getRouteSet(Helper helper){
        return (Graph)maDept[helper.getID()][3];
    }

    public Vector getRouteEdgeSet(Helper helper){
        Graph routeSet=getRouteSet(helper);
        return routeSet.getEdgeSet();
    }

    public synchronized void addEdge(Helper helper,Edge edge){
        Graph routeSet=this.getRouteSet(helper);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(helper,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Helper helper,Node node){
        Graph routeSet=this.getRouteSet(helper);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(helper,routeNodeSet,routeEdgeSet);
    }

    public synchronized void updateMACenter(Helper helper){
        //mutualAssistantCenter
        //maCenter[supply][2]: source(Node)
        //maCenter[supply][3]: icpSet(Vector)
        //maCenter[supply][4]: maPath(Graph)
        //maCenter[supply][7]: maCost > source to supply

        MAManager maMr=helper.getMAManger();
        Center center=maMr.getCenter();
        Node supply=maMr.getSupply();

        center.setSource(supply,helper.getSource());
```

MAManager.java

```
            center.setICPSet(supply,interfaceNodes);
            center.setMAPath(supply,helper.getMAPath());
            center.setMACost(supply,helper.getMACost());
        }
    }
```

Roamer.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;
import emnet.algorithm.GraphAlgorithm;
import emnet.graph.Graph;

public class Roamer extends Thread{
    Node supply;
    int id;
    Center center;
    boolean roamerFinish;

    Node currNode,preNode,myNode;
    Edge preEdge,myEdge;
    Graph graph,routeSet;
    Vector routeNodeSet,routeEdgeSet;

    public Roamer(int id,Node supply,Center center){
        super(""+id);
        this.id=id;
        this.supply=supply;
        this.center=center;
        roamerFinish=false;
        graph=center.getGraph();
        int nodeNum=graph.getNodeSet().size();
        int edgeNum=graph.getEdgeSet().size();

        //roamerCenter init
        //roamerCenter[roamer][0]: myNode(Node)
        //roamerCenter[roamer][1]: myEdge(Edge)
        //roamerCenter[roamer][2]: currNode(Node)
        //roamerCenter[roamer][3]: routeSet(Graph)

        Node dummyNode=this.center.getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,supply,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        this.graph.addNode(dummyNode);
```

Roamer.java

```
        this.graph.addEdge(dummyEdge);

        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);
        this.center.setRouteSet(this,routeNodeSet,routeEdgeSet);

        this.center.setMyNode(this,dummyNode);
        this.center.setMyEdge(this,dummyEdge);
        this.center.setCurrNode(this,supply);

        //nodeCenter init
        //nodeCenter[node][0]: occupy(Boolean)
        //nodeCenter[node][1]: distance(Double)
        //nodeCenter[node][2]: preNode(Node)
        //nodeCenter[node][3]: preEdge(Edge)
        //nodeCenter[node][4]: visitorSequence(Vector)

        this.center.setDistance(supply,0.0);
        this.center.setPreNode(supply,dummyNode);
        this.center.setPreEdge(supply,dummyEdge);
        this.center.addViditor(supply,this);
    }

    public int getID(){
        return this.id;
    }

    public Node getSupply(){
        return this.supply;
    }

    public Center getCenter(){
        return this.center;
    }

    public void run(){

        roaming:
        while(!center.isFinished() && !roamerFinish){
            //take the key to have the right to run
```

Roamer.java

```
center.takeKey(this);

if(center.isFinished()){
    center.putKey(this);
    break roaming;
}

//test currNode is visited or not
if(!center.getCurrNode(this).isVisited()){
    //currNode is not visited
    center.getCurrNode(this).visit();
    updateVisitorSequence(0,this,center.getCurrNode(this));

    //update myEdge, myNode, currNode, preNode
    //update currNode distance by myNode & myEdge
    //update routeSet (myNode, myEdge)
    //set fast edge
    center.addNode(this,center.getCurrNode(this));
    center.addEdge(this,center.getMyEdge(this));

    center.setPreNode(center.getCurrNode(this),center.getMyNode(this));
    center.setPreEdge(center.getCurrNode(this),center.getMyEdge(this));
    double distance=center.getDistance(center.getMyNode(this))+center.getMyEdge(this).getWeight();
    center.setDistance(center.getCurrNode(this),distance);
    center.getMyEdge(this).setTestEdge();
    center.setMyNode(this,center.getCurrNode(this));

    dijkstra();
}else{
    //currNode is visited
    //Comparison: change to new location to find myEdge to find currNode, and update to myNode
    Vector visitorSequence=center.getVisitorSequence(center.getCurrNode(this));
    if(!visitorSequence.contains(this)){
        //comparison
        double currDistance=center.getDistance(center.getCurrNode(this));
        double myDistance=center.getDistance(center.getMyNode(this));
        if(myDistance+center.getMyEdge(this).getWeight()<currDistance){
            //closer! subtree finding!
            //1. find out who the last roamer is
            Roamer lastRoamer=center.lastVisitor(center.getCurrNode(this));

            //2. update visitor sequence [finish condition]
```

Roamer.java

```
                updateVisitorSequence(1,this,center.getCurrNode(this));

                //3. find out where the last roamer is now
                Node lastRoamerLocation=center.getCurrNode(lastRoamer);

                //4. remove last roamer's subtree
                Graph lastRoamerRouteSet=center.getRouteSet(lastRoamer);
                Graph
subtree=GraphAlgorithm.getSubtreeWithCertainNode(lastRoamerRouteSet,center.getCurrNode(this),center.getPreEd
ge(center.getCurrNode(this)));

                center.removeSubtree(lastRoamer,subtree);
                center.removeEdge(lastRoamer,center.getPreEdge(center.getCurrNode(this)));
                center.getPreEdge(center.getCurrNode(this)).setNeutralEdge();

                //5. add myEdge and the subtree to current roamer's RouteSet
                center.setPreEdge(center.getCurrNode(this),center.getMyEdge(this));
                center.setPreNode(center.getCurrNode(this),center.getMyNode(this));
                center.addSubtree(this,subtree);
                center.addEdge(this,center.getMyEdge(this));
                center.getMyEdge(this).setTestEdge();

                //6. relocate last roamer's place to preNode to prevent missing, relocate current roamer
using dijkstra
                center.setCurrNode(lastRoamer,center.getPreNode(center.getCurrNode(this)));

                //7. update the distance in the subtree
                Vector subtreeNodeSet=subtree.getNodeSet();
                Node tempNode;
                double saving;
                for(int i=0;i<subtreeNodeSet.size();i++){
                    tempNode=(Node)subtreeNodeSet.elementAt(i);

saving=center.getDistance(center.getCurrNode(this))-(center.getDistance(center.getMyNode(this))+center.getMyEdg
e(this).getWeight());
                    center.setDistance(tempNode,center.getDistance(tempNode)-saving);
                }

                dijkstra();
            }else{
                //not closer, remove myEdge from routeEdgeSet, find new myEdge
                //update visitor sequence: downstream subree
```

```
                        updateVisitorSequence(1,this,center.getCurrNode(this));
                        center.removeEdge(this,center.getMyEdge(this));

                        dijkstra();
                    }
                }else{
                    //I visited this node before
                    dijkstra();
                }
            }
            center.putKey(this);

            try{
                sleep(1);
            }catch(InterruptedException ex){
                //roamer cannot sleep
            }

        }
        //roamer finished his job!
    }

    void dijkstra(){
        //find out the best incident edge
        //define new myEdge & currNode
        Graph tempRouteSet=(Graph)center.getRouteSet(this);
        Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();

        Vector incidentEdgeSet=GraphAlgorithm.getIncidentEdgeSet(this,tempRouteNodeSet);
        Vector exclusiveIncidentEdgeSet=new Vector();
        for(int i=0;i<incidentEdgeSet.size();i++){
            if(!center.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
                exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
            }
        }

        Edge tempEdge;
        Node n1,n2;
        if(exclusiveIncidentEdgeSet.size()>1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            center.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
```

Roamer.java

```
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                center.setMyNode(this,n1);
                center.setCurrNode(this,n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                center.setMyNode(this,n2);
                center.setCurrNode(this,n1);
            }else{
                //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }

            double min=center.getDistance(center.getMyNode(this))+center.getMyEdge(this).getWeight();
            for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    if(center.getDistance(n1)+tempEdge.getWeight()<min){
                        min=center.getDistance(n1)+tempEdge.getWeight();
                        center.setMyEdge(this,tempEdge);
                        center.setMyNode(this,n1);
                        center.setCurrNode(this,n2);
                    }
                }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                    if(center.getDistance(n2)+tempEdge.getWeight()<min){
                        min=center.getDistance(n2)+tempEdge.getWeight();
                        center.setMyEdge(this,tempEdge);
                        center.setMyNode(this,n2);
                        center.setCurrNode(this,n1);
                    }
                }else{
                    //dijkstra error 2: incident edge error!
                }
            }
        }else if(exclusiveIncidentEdgeSet.size()==1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            center.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                center.setMyNode(this,n1);
                center.setCurrNode(this,n2);
```

Roamer.java

```
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                center.setMyNode(this,n2);
                center.setCurrNode(this,n1);
            }else{
                //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }
        }else{
            roamerFinish=true;
            //renders all nodes visited
            Vector nodeSet=graph.getNodeSet();
            Node tempNode;
            for(int i=0;i<nodeSet.size();i++){
                tempNode=(Node)nodeSet.elementAt(i);
                center.addViditor(tempNode,this);
            }
        }
    }

    void updateVisitorSequence(int sort,Roamer roamer,Node currNode){
        //roamer to be updated....
        //selection choice:
        //case 0: currNode is never visited
        //case 1: currNode is visited

        Vector myVisitorSequence=center.getVisitorSequence(center.getMyNode(this));
        Graph routeSet,subtree;
        Edge preEdge;
        Vector subtreeNodeSet,tempSubtreeNodeVisitorSequence;
        Node tempSubtreeNode,tempPreNode;
        Roamer lastRoamer,tempRoamer;

        switch(sort){
            case 0:
                //case 0: currNode is never visited
                center.addViditor(currNode,roamer);
                break;

            case 1:
                //case 1: currNode is visited
                //add visiotrs to downstream: subtree
                //subtree of currNode
                lastRoamer=center.lastVisitor(currNode);
```

Roamer.java

```
                routeSet=center.getRouteSet(lastRoamer);
                preEdge=center.getPreEdge(currNode);
                tempPreNode=center.getPreNode(currNode);
                subtree=GraphAlgorithm.getSubtreeWithCertainNode(routeSet,currNode,preEdge);
                subtreeNodeSet=subtree.getNodeSet();

                for(int i=0;i<myVisitorSequence.size();i++){
                    tempRoamer=(Roamer)myVisitorSequence.elementAt(i);
                    for(int j=0;j<subtreeNodeSet.size();j++){
                        tempSubtreeNode=(Node)subtreeNodeSet.elementAt(j);
                        tempSubtreeNodeVisitorSequence=center.getVisitorSequence(tempSubtreeNode);
                        if(!tempSubtreeNodeVisitorSequence.contains(tempRoamer))
                            center.addViditor(tempSubtreeNode,tempRoamer);
                    }
                }
                break;
            }
            //update visitorSequence finish!
        }
    }
```

Walker.java

```
package emnet.thread;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;

public class Walker extends Thread{
    double territoryMergeCost;

    Helper helper;
    MAManager maMr;
    Graph territory;
    int demandNum;

    boolean finish;

    public Walker(Helper helper,Node start){
        territoryMergeCost=0.0;

        this.helper=helper;
        maMr=helper.getMAManger();
        territory=maMr.getTerritory();
        demandNum=territory.getDemandNodeNum();

        int edgeNum=maMr.getCenter().getGraph().getEdgeSet().size();
        Node dummyNode=maMr.getCenter().getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        maMr.getGraph().addNode(dummyNode);
        maMr.getGraph().addEdge(dummyEdge);

        //helper init, set to maMr
        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);

        helper.setRouteSet(routeNodeSet,routeEdgeSet);
```

Walker.java

```
        helper.setMyNode(dummyNode);
        helper.setMyEdge(dummyEdge);
        helper.setCurrNode(start);

        helper.setCurrCost(helper.getMyNode(),0.0);

        helper.setPreNode(start,dummyNode);
        helper.setPreEdge(start,dummyEdge);

        finish=false;
    }

    public double getTerritoryMergeCost(){
        return territoryMergeCost;
    }

    public void run(){

        walking:
        while(!finish){
            if(helper.getCurrNode().isDemand()){

                updateFinish();

                helper.addNode(helper.getCurrNode());
                helper.addEdge(helper.getMyEdge());

                helper.setPreNode(helper.getCurrNode(),helper.getMyNode());
                helper.setPreEdge(helper.getCurrNode(),helper.getMyEdge());

                double currCost=helper.getCurrCost(helper.getMyNode())+helper.getMyEdge().getWeight();
                helper.setCurrCost(helper.getCurrNode(),currCost);

                territoryMergeCost=territoryMergeCost+currCost;

                if(!finish){
                    helper.setMyNode(helper.getCurrNode());
                    dijkstra();
                }else{
                    break walking;
                }
            }else{
```

Walker.java

```
                helper.addNode(helper.getCurrNode());
                helper.addEdge(helper.getMyEdge());

                helper.setPreNode(helper.getCurrNode(),helper.getMyNode());
                helper.setPreEdge(helper.getCurrNode(),helper.getMyEdge());

                double currCost=helper.getCurrCost(helper.getMyNode())+helper.getMyEdge().getWeight();
                helper.setCurrCost(helper.getCurrNode(),currCost);

                helper.setMyNode(helper.getCurrNode());
                dijkstra();
            }
        }
    }

    void updateFinish(){
        demandNum--;

        if(demandNum==0)
            finish=true;
    }

    public boolean isFinish(){
        return finish;
    }

    void dijkstra(){
        Graph tempRouteSet=(Graph)helper.getRouteSet();
        Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();
        Vector incidentEdgeSet=getIncidentEdgeSet(territory,tempRouteNodeSet);
        Vector exclusiveIncidentEdgeSet=new Vector();
        for(int i=0;i<incidentEdgeSet.size();i++){
            if(!helper.getRouteSet().hasEdge((Edge)incidentEdgeSet.elementAt(i))){
                exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
            }
        }

        Edge tempEdge;
        Node n1,n2;
        if(exclusiveIncidentEdgeSet.size()>1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            helper.setMyEdge(tempEdge);
```

Walker.java

```
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                helper.setMyNode(n1);
                helper.setCurrNode(n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                helper.setMyNode(n2);
                helper.setCurrNode(n1);
            }
            double min=helper.getCurrCost(helper.getMyNode())+helper.getMyEdge().getWeight();
            for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    if(helper.getCurrCost(n1)+tempEdge.getWeight()<min){
                        min=helper.getCurrCost(n1)+tempEdge.getWeight();
                        helper.setMyEdge(tempEdge);
                        helper.setMyNode(n1);
                        helper.setCurrNode(n2);
                    }
                }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                    if(helper.getCurrCost(n2)+tempEdge.getWeight()<min){
                        min=helper.getCurrCost(n2)+tempEdge.getWeight();
                        helper.setMyEdge(tempEdge);
                        helper.setMyNode(n2);
                        helper.setCurrNode(n1);
                    }
                }
            }
        }else if(exclusiveIncidentEdgeSet.size()==1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            helper.setMyEdge(tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                helper.setMyNode(n1);
                helper.setCurrNode(n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                helper.setMyNode(n2);
                helper.setCurrNode(n1);
            }
```

Walker.java

```
            }else{
                finish=true;
                //renders all nodes visited
            }
        }

        Vector getIncidentEdgeSet(Graph usableGraph,Vector routeNodeSet){
            Edge myEdge=helper.getMyEdge();

            Vector incidentEdges=new Vector();
            Edge tempEdge;
            Node n1,n2;

            for(int i=0;i<routeNodeSet.size();i++){
                Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)routeNodeSet.elementAt(i));
                for(int j=0;j<tempEdgeSet.size();j++){
                    tempEdge=(Edge)tempEdgeSet.elementAt(j);
                    if(!incidentEdges.contains(tempEdge))
                        incidentEdges.addElement(tempEdge);

                    n1=tempEdge.getN1();
                    n2=tempEdge.getN2();
                    if(routeNodeSet.contains(n1) && routeNodeSet.contains(n2))
                        incidentEdges.removeElement(tempEdge);
                    if(n1==maMr.getCenter().getDummyNode() || n2==maMr.getCenter().getDummyNode())
                        incidentEdges.removeElement(tempEdge);
                }
            }
            if(incidentEdges.contains(myEdge))
                incidentEdges.removeElement(myEdge);

            return incidentEdges;
        }
    }
```

App.java

```
package emnet;

import java.awt.Toolkit;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Dimension;

public class App {
  boolean packFrame = false;

  /**
   * Construct and show the application.
   */
  public App() {
    Frame frame = new Frame();
    // Validate frames that have preset sizes
    // Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }

    // Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation( (screenSize.width - frameSize.width) / 2,
                       (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }

  /**
   * Application entry point.
   *
```

App.java

```
     * @param args String[]
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                }
                catch (Exception exception) {
                    exception.printStackTrace();
                }

                //read in node & edge data
                //add to table
                //picture the map
                //frame.add table & map
                //set supply & demand nodes
                //response in map
                //after press "start" button, new ant & antCommunicationCenter
                //show fast routes, detour routes, mutual assistant routes
                //show evaluation indices
                //input expert acceptable time
                //illustrate radar digram and standardized evaluation value

                new App();
            }
        });
    }
}
```

Frame_AboutBox.java

```
package emnet;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Frame_AboutBox
      extends JDialog implements ActionListener {
  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton button1 = new JButton();
  JLabel imageLabel = new JLabel();
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  ImageIcon image1 = new ImageIcon();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "Earthquake Mitigation Network Design";
  String version = "version 1.0, percy.itt.nctu.tw";
  String copyright = "Copyright (c) 2006";
  String comments = "Decision Making Tool for Network Design";

  public Frame_AboutBox(Frame parent) {
    super(parent);
    try {
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      jbInit();
    }
    catch (Exception exception) {
      exception.printStackTrace();
    }
  }

  public Frame_AboutBox() {
```

Frame_AboutBox.java

```
    this(null);
  }

  /**
   * Component initialization.
   *
   * @throws java.lang.Exception
   */
  private void jbInit() throws Exception {
    image1 = new ImageIcon(emnet.Frame.class.getResource("about.png"));
    imageLabel.setIcon(image1);
    setTitle("About");
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
    button1.setText("OK");
    button1.addActionListener(this);
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
    setResizable(true);
  }

  /**
```

Frame_AboutBox.java

```
     * Close the dialog on a button event.
     *
     * @param actionEvent ActionEvent
     */
    public void actionPerformed(ActionEvent actionEvent) {
      if (actionEvent.getSource() == button1) {
        dispose();
      }
    }
}
```

Frame_TermBox.java

```
package emnet;

import javax.swing.JDialog;
import java.awt.GridLayout;
import javax.swing.JLabel;
import java.awt.Dimension;

public class Frame_TermBox extends JDialog{

    JLabel
        label_eva=new JLabel("    E: Overall Evaluation"),
        label_ld=new JLabel("    LD: Longest Detour Cost"),
        label_amac=new JLabel("    AMAC: Average Mutal Assistance Cost"),
        label_nc=new JLabel("    NC: Network Cost"),
        label_atc=new JLabel("    ATC: Average Travel Cost"),
        label_mtc=new JLabel("    MTC: Maximum Travel Cost");

    public Frame_TermBox(Frame parent){
        super(parent);
        try {
          setDefaultCloseOperation(DISPOSE_ON_CLOSE);
          jbInit();
        }
        catch (Exception exception) {
          exception.printStackTrace();
        }
    }

    void jbInit(){
        Dimension d=new Dimension(250,25);
        this.setTitle("Terminology");
        this.getContentPane().setLayout(new GridLayout(6,1,5,5));
        label_eva.setPreferredSize(d);
        label_ld.setPreferredSize(d);
        label_amac.setPreferredSize(d);
        label_nc.setPreferredSize(d);
        label_atc.setPreferredSize(d);
        label_mtc.setPreferredSize(d);

        this.getContentPane().add(label_atc);
        this.getContentPane().add(label_mtc);
```

```
            this.getContentPane().add(label_ld);
            this.getContentPane().add(label_amac);
            this.getContentPane().add(label_nc);
            this.getContentPane().add(label_eva);

            this.setResizable(false);
        }
}
```

Frame.java

```
package emnet;

import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JLabel;

import emnet.gui.Map;
import emnet.graph.Graph;
import java.util.Vector;
import java.io.IOException;
import emnet.io.IOGraph;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import javax.swing.JTabbedPane;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JSplitPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.JTable;
import javax.swing.JScrollPane;
import emnet.graph.Node;
import emnet.graph.Edge;
import emnet.thread.Center;
import java.text.DecimalFormat;

public class Frame
    extends JFrame {
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  JMenuBar jMenuBar1 = new JMenuBar();
  JMenu jMenuFile = new JMenu();
  JMenuItem jMenuFileExit = new JMenuItem();
  JMenu jMenuHelp = new JMenu();
  JMenuItem jMenuHelpAbout = new JMenuItem();
  JMenuItem jMenuHelpTerm=new JMenuItem();
```

Frame.java

```
JPanel panel_gridbag=new JPanel(new GridLayout(4,1,2,2));
JPanel panel_setting=new JPanel(new BorderLayout());

JLabel label_dir=new JLabel("common dir: ");
JTextField txt_dir=new JTextField();
JLabel label_node=new JLabel("node file: ");
JTextField txt_node=new JTextField();
JLabel label_edge=new JLabel("edge file: ");
JTextField txt_edge=new JTextField();
JLabel label_dataIn=new JLabel("2econ?");

DefaultTableModel nodeTableModel,edgeTableModel;
JTable
    nodeTable=new JTable(nodeTableModel),
    edgeTable=new JTable(edgeTableModel);

JLabel seperator=new JLabel("[ no graph ]",SwingConstants.CENTER);

JRadioButton radio_supply=new JRadioButton("supply");
JRadioButton radio_demand=new JRadioButton("demand");
JRadioButton radio_neutral=new JRadioButton("neutral");
ButtonGroup radioGroup=new ButtonGroup();
JButton button_run=new JButton("run");

JTextField
    txt_atc_low=new JTextField(),txt_mtc_low=new JTextField(),
    txt_ld_low=new JTextField(),txt_amac_low=new JTextField(),
    txt_nc_low=new JTextField(),

    txt_atc_up=new JTextField(),txt_mtc_up=new JTextField(),
    txt_ld_up=new JTextField(),txt_amac_up=new JTextField(),
    txt_nc_up=new JTextField();

JLabel
    atc_output=new JLabel("0.0       "),mtc_output=new JLabel("0.0       "),
    ld_output=new JLabel("0.0       "),amac_output=new JLabel("0.0       "),
    nc_output=new JLabel("0.0       "),e_output=new JLabel("[ pls fill \"expert\" tab ]    ");

JLabel save_dir=new JLabel();

Map map=new Map();
```

Frame.java

```
    Graph graph;
    Center center;
    DecimalFormat myFormatter=new DecimalFormat("###,###.##");

    public Frame() {
      try {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        jbInit();
      }
      catch (Exception exception) {
        exception.printStackTrace();
      }
    }

    //Component initialization
    private void jbInit() throws Exception {
      contentPane = (JPanel) getContentPane();
      contentPane.setLayout(borderLayout1);
      setSize(new Dimension(600, 700));
      setTitle(":: EMNet 2006 :: ");
      jMenuFile.setText("File");
      jMenuFileExit.setText("Exit");
      jMenuFileExit.addActionListener(new Frame_jMenuFileExit_ActionAdapter(this));
      jMenuHelp.setText("Help");
      jMenuHelpAbout.setText("About");
      jMenuHelpAbout.addActionListener(new Frame_jMenuHelpAbout_ActionAdapter(this));
      jMenuHelpTerm.setText("Term");
      jMenuHelpTerm.addActionListener(new Frame_jMenuHelpTerm_ActionAdapter(this));
      jMenuBar1.add(jMenuFile);
      jMenuFile.add(jMenuFileExit);
      jMenuBar1.add(jMenuHelp);
      jMenuHelp.add(jMenuHelpAbout);
      jMenuHelp.add(jMenuHelpTerm);
      setJMenuBar(jMenuBar1);

      Dimension big=new Dimension(146,20),big2=new Dimension(185,20),med=new
Dimension(100,20),small=new Dimension(65,20),xs=new Dimension(30,20);

      JPanel panel_bottom=new JPanel(new BorderLayout());
      JPanel panel_left=new JPanel(new BorderLayout());

      panel_gridbag.setPreferredSize(new Dimension(275,120));
```

Frame.java

```
label_dir.setHorizontalAlignment(SwingConstants.RIGHT);
label_dir.setPreferredSize(med);
txt_dir.setPreferredSize(big2);
txt_dir.setText("/Users/percyhou/Desktop/graphFiles");
JPanel gridBag1=new JPanel(new GridBagLayout());
gridBag1.add(label_dir);
gridBag1.add(txt_dir);

label_node.setHorizontalAlignment(SwingConstants.RIGHT);
label_node.setPreferredSize(med);
txt_node.setPreferredSize(big2);
txt_node.setText("node_grid.txt");
JPanel gridBag2=new JPanel(new GridBagLayout());
gridBag2.add(label_node);
gridBag2.add(txt_node);

label_edge.setHorizontalAlignment(SwingConstants.RIGHT);
label_edge.setPreferredSize(med);
txt_edge.setPreferredSize(big2);
txt_edge.setText("edge_grid.txt");
JPanel gridBag3=new JPanel(new GridBagLayout());
gridBag3.add(label_edge);
gridBag3.add(txt_edge);

label_dataIn.setHorizontalAlignment(SwingConstants.RIGHT);
label_dataIn.setPreferredSize(med);
JButton button_import=new JButton("import");
button_import.addActionListener(new Frame_button_import_ActionAdapter(this));

JPanel panel_import=new JPanel(new BorderLayout());
panel_import.add(label_dataIn,BorderLayout.WEST);
panel_import.add(button_import,BorderLayout.EAST);

panel_gridbag.add(gridBag1);
panel_gridbag.add(gridBag2);
panel_gridbag.add(gridBag3);
panel_gridbag.add(panel_import);

panel_left.add(panel_gridbag,BorderLayout.NORTH);

JTabbedPane jtabbedPane=new JTabbedPane();
```

Frame.java

```
jtabbedPane.setPreferredSize(new Dimension(300,300));
JScrollPane tab_node=new JScrollPane(nodeTable);
JScrollPane tab_edge=new JScrollPane(edgeTable);
jtabbedPane.add(tab_node,"node");
jtabbedPane.add(tab_edge,"edge");

panel_left.add(jtabbedPane,BorderLayout.CENTER);
panel_bottom.add(panel_left,BorderLayout.WEST);

JPanel panel_right=new JPanel(new BorderLayout());

//radio buttons: supply, demand, neutral
panel_setting.setPreferredSize(new Dimension(300,120));

radio_supply.addActionListener(new Frame_radio_ActionAdapter(this));
radio_demand.addActionListener(new Frame_radio_ActionAdapter(this));
radio_neutral.addActionListener(new Frame_radio_ActionAdapter(this));
button_run.addActionListener(new Frame_button_run_ActionAdapter(this));

JPanel panel_radio=new JPanel(new GridBagLayout());
radioGroup.add(radio_supply);
radioGroup.add(radio_demand);
radioGroup.add(radio_neutral);
panel_radio.add(radio_supply);
panel_radio.add(radio_demand);
panel_radio.add(radio_neutral);
panel_radio.add(button_run);

panel_setting.add(panel_radio,BorderLayout.NORTH);

//expert bounds
JTabbedPane jtabbedPane_right=new JTabbedPane();
jtabbedPane_right.setPreferredSize(new Dimension(300,200));
JPanel tab_index=new JPanel(new BorderLayout());
JPanel panel_index=new JPanel(new GridLayout(9,1,2,2));
JPanel tab_result=new JPanel(new GridLayout(9,1,2,2));
tab_index.add(panel_index,BorderLayout.NORTH);
jtabbedPane_right.add(tab_result,"result");
jtabbedPane_right.add(tab_index,"expert");

JPanel panel_expert0=new JPanel(new BorderLayout());
JPanel panel_expert1=new JPanel(new GridBagLayout());
```

Frame.java

```
JPanel panel_expert2=new JPanel(new GridBagLayout());
JPanel panel_expert3=new JPanel(new GridBagLayout());
JPanel panel_expert4=new JPanel(new GridBagLayout());
JPanel panel_expert5=new JPanel(new GridBagLayout());
JPanel panel_expert6=new JPanel(new BorderLayout());

JLabel
    label_mtc=new JLabel(" MTC: ",SwingConstants.RIGHT),
    label_atc=new JLabel(" ATC: ",SwingConstants.RIGHT),
    label_ld=new JLabel(" LD: ",SwingConstants.RIGHT),
    label_amac=new JLabel(" AMAC: ",SwingConstants.RIGHT),
    label_nc=new JLabel(" NC: ",SwingConstants.RIGHT);

JLabel
    label_atc_unit=new JLabel(" m",SwingConstants.LEFT),
    label_mtc_unit=new JLabel(" m",SwingConstants.LEFT),
    label_ld_unit=new JLabel(" m",SwingConstants.LEFT),
    label_amac_unit=new JLabel(" m",SwingConstants.LEFT),
    label_nc_unit=new JLabel(" m",SwingConstants.LEFT);

label_atc.setPreferredSize(small);
label_mtc.setPreferredSize(small);
label_ld.setPreferredSize(small);
label_amac.setPreferredSize(small);
label_nc.setPreferredSize(small);

txt_atc_low.setPreferredSize(small);
txt_mtc_low.setPreferredSize(small);
txt_ld_low.setPreferredSize(small);
txt_amac_low.setPreferredSize(small);
txt_nc_low.setPreferredSize(small);

txt_atc_up.setPreferredSize(small);
txt_mtc_up.setPreferredSize(small);
txt_ld_up.setPreferredSize(small);
txt_amac_up.setPreferredSize(small);
txt_nc_up.setPreferredSize(small);

label_atc_unit.setPreferredSize(xs);
label_mtc_unit.setPreferredSize(xs);
label_ld_unit.setPreferredSize(xs);
label_amac_unit.setPreferredSize(xs);
```

Frame.java

```
label_nc_unit.setPreferredSize(xs);

JButton button_set=new JButton("set");
button_set.addActionListener(new Frame_button_set_ActionAdapter(this));

panel_expert0.add(new JLabel(" acceptable range:"),BorderLayout.WEST);

panel_expert1.add(label_atc);
panel_expert1.add(txt_atc_low);
panel_expert1.add(new JLabel(" ~ "));
panel_expert1.add(txt_atc_up);

panel_expert2.add(label_mtc);
panel_expert2.add(txt_mtc_low);
panel_expert2.add(new JLabel(" ~ "));
panel_expert2.add(txt_mtc_up);

panel_expert3.add(label_ld);
panel_expert3.add(txt_ld_low);
panel_expert3.add(new JLabel(" ~ "));
panel_expert3.add(txt_ld_up);

panel_expert4.add(label_amac);
panel_expert4.add(txt_amac_low);
panel_expert4.add(new JLabel(" ~ "));
panel_expert4.add(txt_amac_up);

panel_expert5.add(label_nc);
panel_expert5.add(txt_nc_low);
panel_expert5.add(new JLabel(" ~ "));
panel_expert5.add(txt_nc_up);

panel_expert6.add(button_set,BorderLayout.EAST);

panel_index.add(panel_expert0);
panel_index.add(panel_expert1);
panel_index.add(panel_expert2);
panel_index.add(panel_expert3);
panel_index.add(panel_expert4);
panel_index.add(panel_expert5);
panel_index.add(panel_expert6);
```

Frame.java

```
        JTextField txtField=new JTextField("[indices]",15);
        txtField.setBackground(Color.WHITE);
        txtField.setEditable(false);

        JPanel panel_output0=new JPanel(new GridBagLayout());
        JPanel panel_output3=new JPanel(new GridBagLayout());
//      JPanel panel_output29=new JPanel(new GridBagLayout());
        JPanel panel_output4=new JPanel(new GridBagLayout());
//      JPanel panel_output49=new JPanel(new GridBagLayout());
        JPanel panel_output5=new JPanel(new GridBagLayout());
        JPanel panel_output1=new JPanel(new GridBagLayout());
        JPanel panel_output2=new JPanel(new GridBagLayout());
        JPanel panel_output6=new JPanel(new BorderLayout());

        JLabel
            label_atc1=new JLabel(" ATC: ",SwingConstants.RIGHT),
            label_mtc1=new JLabel(" MTC: ",SwingConstants.RIGHT),
            label_ld1=new JLabel(" LD: ",SwingConstants.RIGHT),
            label_amac1=new JLabel(" AMAC: ",SwingConstants.RIGHT),
            label_nc1=new JLabel(" NC: ",SwingConstants.RIGHT),
            label_e=new JLabel(" E: ",SwingConstants.RIGHT);

        JLabel
            label_atc_unit1=new JLabel(" ",SwingConstants.LEFT),
            label_mtc_unit1=new JLabel(" ",SwingConstants.LEFT),
            label_ld_unit1=new JLabel(" ",SwingConstants.LEFT),
            label_amac_unit1=new JLabel(" ",SwingConstants.LEFT),
            label_nc_unit1=new JLabel(" ",SwingConstants.LEFT),
            label_e_unit1=new JLabel();

        label_atc1.setPreferredSize(small);
        label_mtc1.setPreferredSize(small);
        label_ld1.setPreferredSize(small);
        label_amac1.setPreferredSize(small);
        label_nc1.setPreferredSize(small);
        label_e.setPreferredSize(small);

        atc_output.setPreferredSize(big);
        mtc_output.setPreferredSize(big);
        ld_output.setPreferredSize(big);
        amac_output.setPreferredSize(big);
        nc_output.setPreferredSize(big);
```

```
e_output.setPreferredSize(big);

atc_output.setHorizontalAlignment(SwingConstants.RIGHT);
mtc_output.setHorizontalAlignment(SwingConstants.RIGHT);
ld_output.setHorizontalAlignment(SwingConstants.RIGHT);
amac_output.setHorizontalAlignment(SwingConstants.RIGHT);
nc_output.setHorizontalAlignment(SwingConstants.RIGHT);
e_output.setHorizontalAlignment(SwingConstants.RIGHT);

label_atc_unit1.setPreferredSize(xs);
label_mtc_unit1.setPreferredSize(xs);
label_ld_unit1.setPreferredSize(xs);
label_amac_unit1.setPreferredSize(xs);
label_nc_unit1.setPreferredSize(xs);
label_e_unit1.setPreferredSize(xs);

panel_output0.add(label_e);
panel_output0.add(e_output);
panel_output0.add(label_e_unit1);

panel_output1.add(label_atc1);
panel_output1.add(atc_output);
panel_output1.add(label_atc_unit1);

panel_output2.add(label_mtc1);
panel_output2.add(mtc_output);
panel_output2.add(label_mtc_unit1);

panel_output3.add(label_ld1);
panel_output3.add(ld_output);
panel_output3.add(label_ld_unit1);

panel_output4.add(label_amac1);
panel_output4.add(amac_output);
panel_output4.add(label_amac_unit1);

panel_output5.add(label_nc1);
panel_output5.add(nc_output);
panel_output5.add(label_nc_unit1);

save_dir.setPreferredSize(new Dimension(200,20));
JButton button_save=new JButton("save");
```

Frame.java

```
//      button_save.addActionListener(this);

        panel_output6.add(save_dir,BorderLayout.WEST);
        panel_output6.add(button_save,BorderLayout.EAST);

        tab_result.add(panel_output1);
        tab_result.add(panel_output2);
        tab_result.add(panel_output3);
        tab_result.add(panel_output4);
        tab_result.add(panel_output5);
        tab_result.add(panel_output0);
        tab_result.add(panel_output6);

        panel_setting.add(jtabbedPane_right,BorderLayout.CENTER);
        panel_right.add(panel_setting,BorderLayout.EAST);
        panel_bottom.add(panel_right,BorderLayout.CENTER);
        panel_bottom.add(seperator,BorderLayout.NORTH);

        map.sentFrame(this);

        JSplitPane sp=new JSplitPane(JSplitPane.VERTICAL_SPLIT,map,panel_bottom);
        sp.setDividerSize(8);
        sp.setDividerLocation(270);
        sp.setResizeWeight(0.5);
        sp.setContinuousLayout(true);
        sp.setOneTouchExpandable(true);
        contentPane.add(sp,BorderLayout.CENTER);
    }

    public void setATC(double atc){
        atc_output.setText(myFormatter.format(atc)+"        ");
        txt_atc_low.setText(myFormatter.format(atc*0.8));
        txt_atc_up.setText(myFormatter.format(atc*1.2));
    }

    public void setMTC(double mtc){
        mtc_output.setText(myFormatter.format(mtc)+"        ");
        txt_mtc_low.setText(myFormatter.format(mtc*0.8));
        txt_mtc_up.setText(myFormatter.format(mtc*1.2));
    }

    public void setLD(double ld){
```

Frame.java

```
        ld_output.setText(myFormatter.format(ld)+"        ");
        txt_ld_low.setText(myFormatter.format(ld*0.8));
        txt_ld_up.setText(myFormatter.format(ld*1.2));
    }

    public void setAMAC(double amac){
        amac_output.setText(myFormatter.format(amac)+"        ");
        txt_amac_low.setText(myFormatter.format(amac*0.8));
        txt_amac_up.setText(myFormatter.format(amac*1.2));
    }

    public void setNC(double nc){
        nc_output.setText(myFormatter.format(nc)+"        ");
        txt_nc_low.setText(myFormatter.format(nc*0.8));
        txt_nc_up.setText(myFormatter.format(nc*1.2));
    }

    public void setE(double e){
        e_output.setText(myFormatter.format(e)+"        ");
    }

    void jMenuFileExit_actionPerformed(ActionEvent actionEvent) {
       System.exit(0);
    }

    void jMenuHelpAbout_actionPerformed(ActionEvent actionEvent) {
       Frame_AboutBox dlg = new Frame_AboutBox(this);
       Dimension dlgSize = dlg.getPreferredSize();
       Dimension frmSize = getSize();
       Point loc = getLocation();
       dlg.setLocation( (frmSize.width - dlgSize.width) / 2 + loc.x,
                        (frmSize.height - dlgSize.height) / 2 + loc.y);
       dlg.setModal(true);
       dlg.pack();
       dlg.show();
    }

    void jMenuHelpTerm_actionPerformed(ActionEvent actionEvent) {
       Frame_TermBox dlg = new Frame_TermBox(this);
       Dimension dlgSize = dlg.getPreferredSize();
       Dimension frmSize = getSize();
       Point loc = getLocation();
```

Frame.java

```
    dlg.setLocation( (frmSize.width - dlgSize.width) / 2 + loc.x,
                     (frmSize.height - dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.pack();
    dlg.show();
}

void button_import_actionPerformed(ActionEvent actionEvent){

    setATC(0.0);
    setMTC(0.0);
    setLD(0.0);
    setAMAC(0.0);
    setNC(0.0);
    setE(0.0);

    //read in files: node.txt & edge.txt, assign to field argument
    try {
        IOGraph ioGraph=new IOGraph(txt_dir.getText(),txt_node.getText(),txt_edge.getText());
        this.graph=ioGraph.getGraph();
        Vector nodeSet=graph.getNodeSet();
        Vector edgeSet=graph.getEdgeSet();

        Node tempNode;
        boolean econ=true;
        for(int i=0;i<nodeSet.size();i++){
            tempNode=(Node)nodeSet.elementAt(i);
            if(graph.incidentEdgeSet(tempNode).size()<2){
                econ=false;
            }
        }

        if(econ){
            label_dataIn.setText("2econ!");
        }else{
            label_dataIn.setText("not 2econ!");
        }
    }
    catch (IOException ex) {
        System.out.println("Data Read-In Problem!");
        ex.printStackTrace();
    }
```

Frame.java

```
        save_dir.setText("dir: "+txt_dir.getText());

        map.setGraph(graph);
        this.seperator.setText("[ graph components: "+graph.getNodeSet().size()+" nodes,
"+graph.getEdgeSet().size()+" edges ]");

        this.fleshTable(graph);
        this.repaint();
    }

    void radio_actionPerformed(ActionEvent actionEvent){
        map.nodeSetting(radio_supply.isSelected(),radio_demand.isSelected(),radio_neutral.isSelected());
        this.repaint();
    }

    void button_run_actionPerformed(ActionEvent actionEvent){
        center=new Center(graph,this);
        center.start();

        this.repaint();
    }

    void button_set_actionPerformed(ActionEvent actionEvent){
        double e=1.0;

        double atc=new Double(atc_output.getText()).doubleValue();
        double low_atc=new Double(txt_atc_low.getText()).doubleValue();
        double up_atc=new Double(txt_atc_up.getText()).doubleValue();
        double std_atc=standardN(atc,low_atc,up_atc);

        double mtc=new Double(mtc_output.getText()).doubleValue();
        double low_mtc=new Double(txt_mtc_low.getText()).doubleValue();
        double up_mtc=new Double(txt_mtc_up.getText()).doubleValue();
        double std_mtc=standardN(mtc,low_mtc,up_mtc);

        double amac=new Double(amac_output.getText()).doubleValue();
        double low_amac=new Double(txt_amac_low.getText()).doubleValue();
        double up_amac=new Double(txt_amac_up.getText()).doubleValue();
        double std_amac=standardN(amac,low_amac,up_amac);

        double ld=new Double(ld_output.getText()).doubleValue();
```

Frame.java

```
        double low_ld=new Double(txt_ld_low.getText()).doubleValue();
        double up_ld=new Double(txt_ld_up.getText()).doubleValue();
        double std_ld=standardN(ld,low_ld,up_ld);

        double nc=new Double(nc_output.getText()).doubleValue();
        double low_nc=new Double(txt_nc_low.getText()).doubleValue();
        double up_nc=new Double(txt_nc_up.getText()).doubleValue();
        double std_nc=standardP(nc,low_nc,up_nc);

        e=(1.0/3.0)*((std_atc + std_mtc)/2.0 + (std_ld + std_amac)/2.0 + std_nc);

        this.setE(e);
    }

    //positive
    double standardP(double x,double low,double up){
        if(up<=x){
            return 1.0;
        }else if(low<=x && x<up){
            return (x-low)/(up-low);
        }else{
            return 0.0;
        }
    }

    //negative
    double standardN(double x,double low,double up){
        if(x<low){
            return 1.0;
        }else if(low<=x && x<up){
            return (low-x)/(low-up);
        }else{
            return 0.0;
        }
    }


    public void setSeperator(Graph graph){

        int nodeNum=0,edgeNum=0;
        Vector nodeSet=graph.getNodeSet();
        Vector edgeSet=graph.getEdgeSet();
```

Frame.java

```
        Node tempNode;
        for(int i=0;i<nodeSet.size();i++){
            tempNode=(Node)nodeSet.elementAt(i);
            if(!tempNode.isDummy())
                nodeNum++;
        }

        Edge tempEdge;
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(!tempEdge.isDummyEdge())
                edgeNum++;
        }

        seperator.setText("[ graph components: "+nodeNum+" nodes, "+edgeNum+" edges   |   supply
"+graph.getSupplyNodeNum()+" , demand "+graph.getDemandNodeNum()+" ]");
        this.repaint();
    }

    public Map getMap(){
        return this.map;
    }

    void fleshTable(Graph graph){
        Vector nodeSet=graph.getNodeSet(),edgeSet=graph.getEdgeSet();
        Object nodeData[]=new Object[3];
        DefaultTableModel nodeTableModel=new DefaultTableModel();
        nodeTableModel.addColumn("label");
        nodeTableModel.addColumn("x");
        nodeTableModel.addColumn("y");
        for(int i=0;i<nodeSet.size();i++){
            Node node=(Node)nodeSet.elementAt(i);
            nodeData[0]=""+node.getLabel();
            nodeData[1]=""+node.getX();
            nodeData[2]=""+node.getY();
            nodeTableModel.addRow(nodeData);
        }
        this.nodeTable.setModel(nodeTableModel);

        Object edgeData[]=new Object[3];
        DefaultTableModel edgeTableModel=new DefaultTableModel();
        edgeTableModel.addColumn("label");
```

Frame.java

```
            edgeTableModel.addColumn("n1");
            edgeTableModel.addColumn("n2");
            for(int i=0;i<edgeSet.size();i++){
                Edge edge=(Edge)edgeSet.elementAt(i);
                edgeData[0]=""+edge.getLabel();
                edgeData[1]=""+edge.getN1().getLabel();
                edgeData[2]=""+edge.getN2().getLabel();
                edgeTableModel.addRow(edgeData);
            }
            this.edgeTable.setModel(edgeTableModel);
    }

}

class Frame_jMenuFileExit_ActionAdapter
        implements ActionListener {
    Frame adaptee;

    Frame_jMenuFileExit_ActionAdapter(Frame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuFileExit_actionPerformed(actionEvent);
    }
}

class Frame_jMenuHelpAbout_ActionAdapter
        implements ActionListener {
    Frame adaptee;

    Frame_jMenuHelpAbout_ActionAdapter(Frame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuHelpAbout_actionPerformed(actionEvent);
    }

}

class Frame_jMenuHelpTerm_ActionAdapter
```

Frame.java

```
    implements ActionListener {
  Frame adaptee;

  Frame_jMenuHelpTerm_ActionAdapter(Frame adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent actionEvent) {
    adaptee.jMenuHelpTerm_actionPerformed(actionEvent);
  }

}


class Frame_button_import_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_button_import_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent){
        adaptee.button_import_actionPerformed(actionEvent);
    }
}

class Frame_radio_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_radio_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent){
        adaptee.radio_actionPerformed(actionEvent);
    }
}

class Frame_button_run_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_button_run_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }
```

Frame.java

```
        public void actionPerformed(ActionEvent actionEvent){
            adaptee.button_run_actionPerformed(actionEvent);
        }
    }

    class Frame_button_set_ActionAdapter implements ActionListener{
        Frame adaptee;
        Frame_button_set_ActionAdapter(Frame adaptee){
            this.adaptee=adaptee;
        }

        public void actionPerformed(ActionEvent actionEvent){
            adaptee.button_set_actionPerformed(actionEvent);
        }
    }
```

個人簡歷

侯鵬曦

交通大學交通運輸研究所　博士　（90.09~95.12）
交通大學交通運輸研究所　碩士　（88.09~90.06）
台灣大學土木工程學系　學士　（84.09~88.06）

A. 期刊論文

1. 侯鵬曦、徐淵靜，「Survivable network design model for earthquake disaster」，中國土木水利工程學刊（已接受）。
2. Hsu, Y.C. and Hou, P.H. “Finding dominant links of emergency network with respect to earthquake disaster”, Journal of the Eastern Asia Society for Transportation Studies, 6, pp. 77-90, 2005.
3. 徐淵靜、侯鵬曦，「高雄市防災路網研擬之研究－地理資訊系統之應用」，中華道路，第四十二卷，第二三四期，頁10-20，民國92年。

B. 研討會論文

4. Hou, P.H. “Network planning for earthquake disaster in Kaohsiung City”, 2006 International Conference on Deep Ocean Water and Industrial Development, Kaohsiung, Taiwan, 2006.
5. Hou, P.H., Lien, Barry C.S. and Chen, Johnny J.Y. “ITS-based road network application: Finding shortest time path under traffic signal system”, The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
6. Lien, Barry C.S., Chen, Johnny J.Y. and Hou, P.H. “Advanced timetable query system in Taiwan railway transportation”, The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
7. Lien, Barry C.S., Chen, Johnny J.Y. and Hou, P.H. “Implementation of dynamic navigation system with real-time traffic events using digital audio broadcasting (DAB) as the communication system”, The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
8. 侯鵬曦，「以虛擬身分構築線上購物之安全環境」，2006電子商務與數位生活研討會，台北大學三峽校區，民國95年。
9. 侯鵬曦，「數位生活之險境－數位落差形成原因之初探」，2006電子商務與數位生活研討會，台北大學三峽校區，民國95年。
10. Hou, H.S. and Hou, P.H. “Establishment of Taiwan Hub Port”, Proceedings of the 9th HKSTS Conference, Hong Kong, China, 2004.
11. Hou, H.S. and Hou, P.H. “Integrated Development Project of Ports in Taiwan”, Proceedings of

the 9th HKSTS Conference, Hong Kong, China, 2004.

12. 徐淵靜、侯鵬曦,「路網配置之防震災品質評估」,第十九屆中華民國運輸研討會,台南長榮大學,民國93年。
13. Hsu, Y.C. and Hou, P.H. “Rescue network design with respect to earthquake”, Proceedings of International Symposium on City Planning 2004, Sapporo, Japan, 2004.
14. Hsu, Y.C., Hsia, L.M., Hsu, M.C. and Hou, P.H. “ Urban location characteristics concerned by enterprises: A case in Taipei”, Proceedings of International Symposium on City Planning 2004, Sapporo, Japan, 2004.
15. 徐淵靜、侯鵬曦,「以消費者導向構建電子商店之付費系統」,2004海峽兩岸智慧型運輸系統論文研討會論文集,中國哈爾濱,民國93年。
16. Hou, H.S. and Hou, P.H. “Integrated Planning of Kaohsiung Port”, Proceedings of PACON 2004, Hawaii, USA, 2004.
17. 徐淵靜、侯鵬曦,「防災路網模式構建」,第十八屆中華民國運輸研討會論文集,新竹交通大學,民國92年。
18. Hou, H.S. and Hou, P.H. “Environmental impact of redevelopment of port of Keelung”, Proceedings of PACON 2003, Kaohsiung, Taiwan, 2003.
19. Hou, H.S. and Hou, P.H. “Transportation policy: Case study of Kaohsiung city, Proceedings of the 5th EAST, Fukuoka, Japan, 2003.
20. Hou, H.S. and Hou, P.H. “Offshore zone reclamation: Kaohsiung South Star Plan, Proceedings of PACON 2002, Chiba, Japan, 2002.

C. 研究報告及其他

21. 高齡社會的來臨：為2025年的台灣社會規劃之整合研究「高齡社會之交通與運輸」,國科會專題研究，民國95年。
22. 市區道路防災服務水準之評估II，國科會專題研究，民國94年。
23. 防災路網服務水準之評估模式I，國科會專題研究，民國93年。
24. 都市防災道路系統之研究，國科會專題研究，民國91年。
25. 台北縣交通空氣污染改善減量成效評估計畫，北縣環保局，民國91年。
26. 機場噪音輔助執行項目分析評估，空軍總部，民國91年。