Supplementary Material

OneStageRO.java

```
import ilog.concert.IloException;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;
import java.io.IOException;

public class OneStageRO {

    public static void OneStageRO(double Theta) {
        try {
            IloCplex cplex = new IloCplex();

            double Tmax = ReadData.Tmax;
            int n = ReadData.mNumRequests;
            double b[] = ReadData.b;
            double[][] distance = ReadData.distance;
            double[][] deviation = ReadData.deviation;

            IloNumVar[][] y = new IloNumVar[n][n];
            for(int i=0; i<n; i++) {
                y[i] = cplex.boolVarArray(n);
            }
            IloNumVar[] u = cplex.numVarArray(n-1, 1, n-1);

            //objective
            IloLinearNumExpr obj = cplex.linearNumExpr();
            for(int i=1; i<n; i++) {
                for(int j=0; j<n; j++)
                    if(j!=i)
                        obj.addTerm(b[i],y[i][j]);
            }
            cplex.addMaximize(obj);

            //constraints
            //1
            IloLinearNumExpr constraint, constraint1;
            constraint = cplex.linearNumExpr();
            for(int i=0; i<n; i++) {
                for(int j=0; j<n; j++) {
```

```
            constraint.addTerm(distance[i][j]+Theta*deviation[i][j], y[i][j]);
        }
    }
    cplex.addLe(constraint, Tmax);

    //2
    constraint = cplex.linearNumExpr();
    for(int i=1; i<n; i++)
        constraint.addTerm(1, y[i][0]);
    cplex.addEq(constraint, 1);
    constraint = cplex.linearNumExpr();
    for(int j=1; j<n; j++)
        constraint.addTerm(1, y[0][j]);
    cplex.addEq(constraint, 1);


    //3
    for(int j=1; j<n; j++) {
        constraint = cplex.linearNumExpr();
        constraint1 = cplex.linearNumExpr();
        for(int i=0; i<n; i++) {
            if(i!=j) {
                constraint.addTerm(1, y[i][j]);
                constraint1.addTerm(1, y[j][i]);
            }
        }
        cplex.addEq(constraint, constraint1);
        cplex.addLe(constraint,1);
        cplex.addLe(constraint1, 1);
    }

    //4
    for(int i=1; i<n; i++) {
        for(int j=1; j<n; j++) {
                constraint = cplex.linearNumExpr();
                constraint.addTerm(1, u[i-1]);
                constraint.addTerm(-1, u[j-1]);
                constraint.addTerm(n-1,y[i][j]);
                cplex.addLe(constraint, n-2);
        }
    }

    //solve
    cplex.solve();
```

```
            cplex.end();

        } catch (IloException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws IOException {
        String file = "./data.txt";
        ReadData.read(file);
        OneStageRO(0.1);
    }
}
```

ReadData.java

```
import java.io.*;
import java.util.List;
import java.util.Random;

public class ReadData {
    public static int mNumRequests;
    public static double xPos[];
    public static double yPos[];
    public static double b[];
    public static double deviation[][];
    public static double[][] distance;
    public static final double Tmax = 70;
    public static double Max_dev;

    public static double nominalDis (List<Arc> arcs) {
        double dis = 0;
        for(Arc arc : arcs) {
            int i = arc.head;
            int j = arc.tail;
            dis += distance[i][j];
        }
        return dis;
    }

    public static double robustDis (List<Arc> arcs) {
        double dis = 0;
```

```
        for(Arc arc : arcs) {
            int i = arc.head;
            int j = arc.tail;
            dis += distance[i][j] + deviation[i][j];
        }
        return dis;
    }

    public static double[] getDeviations(List<Arc> arcs, int length) {
        double[] deviations = new double[length];
        int index = 0;
        for(Arc arc : arcs) {
            int i = arc.head;
            int j = arc.tail;
            deviations[index] = deviation[i][j];
            index++;
        }
        return deviations;
    }

    public static void read(String input) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(new File(input)));
        int lineNumber = 0;
        String line = reader.readLine();
        while (line != null) {
            if (line.startsWith(" "))
                line = line.replaceFirst(" +", "");
            parseLine(line, lineNumber);

            line = reader.readLine();
            lineNumber++;
        }
        reader.close();

        Random r = new Random(0);

        for(int i=0; i<mNumRequests; i++)
            for(int j=0; j<mNumRequests; j++) {
                distance[i][j] = Math.sqrt(Math.pow(xPos[i] - xPos[j], 2) + Math.pow(yPos[i] - yPos[j], 2));
                //deviation[i][j] = r.nextDouble()*distance[i][j];
                deviation[i][j] = 0.5*distance[i][j];
            }

        Max_dev = 0;
```

```
            for(int i=0; i<mNumRequests; i++) {
                for (int j = 0; j < mNumRequests; j++) {
                    if (deviation[i][j] > Max_dev) {
                        Max_dev = deviation[i][j];
                    }
                }
            }
        }

        private static void parseLine(String line, int lineNumber) {
            if (lineNumber == 0) {
                mNumRequests = Integer.parseInt(line);
                xPos = new double[mNumRequests];
                yPos = new double[mNumRequests];
                b = new double[mNumRequests];
                deviation = new double[mNumRequests][mNumRequests];
                distance = new double[mNumRequests][mNumRequests];
            } else if (lineNumber <= mNumRequests) {
                String[] values = line.split("\\s+");
                xPos[lineNumber-1] = Double.parseDouble(values[0]);
                yPos[lineNumber-1] = Double.parseDouble(values[1]);
                b[lineNumber-1] = Double.parseDouble(values[2]);
            }
        }
}
```

data.txt

```
20
4.6     7.1     0
5.7     11.4    20
4.4     12.3    20
2.8     14.3    30
3.2     10.30   15
3.5     9.8     15
4.4     8.4     10
7.8     11.0    20
8.8     9.8     20
7.7     8.2     20
6.3     7.9     15
5.4     8.2     10
5.8     6.8     10
6.7     5.8     25
13.8    13.1    40
```

14.1	14.2	40
11.2	13.6	30
9.7	16.4	30
9.5	18.8	50
4.7	16.8	30