Raport Badawczy

Research Report

RB/1/2007

MIP formulation of the adjustment problem for the MST and MHP problems: computational results

J. Kurnatowski, M. Libura

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences

POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ul. Newelska 6

01-447 Warszawa

tel.: (+48) (22) 8373578

fax: (+48) (22) 8372772

Kierownik Pracowni zgłaszający pracę:
prof. dr bab. inż. Krzysztof Kiwiel

Warszawa 2006

MIP formulation of the adjustment problem for the MST and MHP problems: computational results

Jan Kurnatowski
Systems Research Institute
Polish Academy of Sciences
01-447 Warsaw, Poland
E-mail: J.Kurnatowski@wit.edu.pl

Marek Libura
Systems Research Institute
Polish Academy of Sciences
01-447 Warsaw, Poland
E-mail: Marek.Libura@ibspan.waw.pl

Abstract

We consider a pair of optimization problems for a given weighted undirected graph: the minimum spanning tree (MST) problem and its restriction, the minimum Hamiltonian path (MHP) problem. For this pair of problems the adjustment problem consists in finding such minimum norm perturbations of weights of edges, which guarantee that the set of optimal solutions of the MST problem in the perturbed graph contains a Hamiltonian path.

In this paper we consider a mixed integer programming (MIP) formulation of the adjustment problem and describe computational results obtained for randomly generated graphs.

1 Introduction

The adjustment problem described in [1] is formulated for a pair of optimization problems: an initial optimization problem with linear objective function

$$v(c, X) = \max\{c^T x : x \in X\}, \tag{1}$$

where $c \in \mathbb{R}^n$ and $X \subseteq \mathbb{R}^n$, and its restriction for a given subset $F \subseteq X$ of feasible solutions:

$$\max\{c^T x : x \in F\}. \tag{2}$$

Given such a pair of problems one wants to adjust the objective function coefficients in problem (1), such that an optimal solution of the perturbed problem is also a feasible solution for the restriction (2). The adjustment problem seeks among admissible perturbations, one that is least costly according to some given norm.

Let $\Delta \subseteq \mathbb{R}^n$ be the set of all admissible perturbations of the original vector of weights c, and let $\|\delta\|$ denotes a norm of $\delta \in \mathbb{R}^n$.

The adjustment problem related to F and Δ is formally stated as follows:

$$a(F, \Delta) = \min\{\|\delta\| : v(c + \delta, X) = v(c + \delta, F), \delta \in \Delta\}. \tag{3}$$

Given sets $\Delta \subseteq \mathbb{R}^n$ and $F \subseteq X$, the optimal value $a(F, \Delta)$ of the problem (3) is called the adjustment cost with respect to F and Δ. In this paper we will consider only l_1 norm, i.e., for $\delta = \delta^+ - \delta^-$, $\delta^+, \delta^- \in \mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geq 0\}$, $\|\delta\| = \mathbf{1}^T\delta^+ - \mathbf{1}^T\delta^-$, where $\mathbf{1} \in \mathbb{R}^n$ is a vector of ones. Moreover, we will take $\Delta = \mathbb{R}^n$.

In the following we assume that the initial optimization problem (1) is a linear programming problem:

$$\max c^T x \tag{4a}$$

$$A'x \leq b' \tag{4b}$$

$$A''x = b'' \tag{4c}$$

$$x \geq 0, \tag{4d}$$

where $A' \in \mathbb{R}^{m' \times n}$, $A'' \in \mathbb{R}^{m'' \times n}$, $b' \in \mathbb{R}^{m'}$ and $b'' \in \mathbb{R}^{m''}$.

A restriction of this problem will be defined by adding new linear constraints

$$Cx \leq d, \tag{5}$$

where $C \in \mathbb{R}^{p \times n}$, $d \in \mathbb{R}^p$, and requiring that feasible solutions belong to the set $\mathbb{B}^n = \{0, 1\}^n$. Thus, the restricted problem (2) considered in the paper has the form:

$$\begin{aligned} &\max c^T x \\ &A'x \leq b' \\ &A''x = b'' \\ &Cx \leq d \\ &x \in \mathbb{B}^n. \end{aligned} \tag{6}$$

In [1] it is shown that the adjustment problem for a pair of problems (4) and (6) and $\|\delta\| = l_1$, $\Delta = \mathbb{R}^n$, can be stated as follows:

$$\min(\mathbf{1}^T\delta^+ - \mathbf{1}^T\delta^-) \tag{7a}$$

$$\begin{cases} (A')^T y' - \delta^+ + \delta^- \geq c \\ (A'')^T y'' - \delta^+ + \delta^- \geq c \\ y' \geq 0, y'' \in \mathbb{R}^{m''} \end{cases} \tag{7b}$$

$$(b')^T y' + (b'')^T y'' - c^T x - (\delta^+)^T x + (\delta^-)^T x = 0 \tag{7c}$$

$$\begin{cases} A'x \leq b' \\ A''x = b'' \\ Cx \leq d \\ x \in \mathbb{B}^n \end{cases} \tag{7d}$$

$$\delta^+, \delta^- \geq 0. \tag{7e}$$

In the above problem the constraints (7b) and variables y', y'', correspond to the dual problem for linear program (4), whereas constraints (7d) and variables x correspond to the restricted problem (6).

The nonlinear constraint (7c) states that the value of the objective function in the perturbed initial linear program (4) is equal to the value of the objective function in its dual. To solve problem (7) with standard optimization packages for MIP it is necessary to linearize the nonlinear terms $(\delta^+)^T x$ and $(\delta^-)^T x$. In the next section we describe this step and give all details for MIP formulation of the adjustment problem for a pair of problems: the minimum spanning tree (MST) problem, regarded as the initial optimization problem (4), and the minimum Hamiltonian path problem – as its restriction (6).

2 MST, MHP and the adjustment problems

Let $G = (V, E, c)$ be a connected weighted graph, where $V = \{1, \ldots, n\}$, $E \subseteq \{\{i,j\} : i,j \in V\}$, $c \in \mathbb{R}^{|E|}$. Our goal will be to find such perturbations $\delta \in \mathbb{R}^{|E|}$ of the weight vector c, that some minimum spanning tree in the perturbed graph is a Hamiltonian path in this graph. First, we will formulate the minimum spanning tree problem in G as the initial linear program (4). Then, we describe additional constraints to get the minimum Hamiltonian path problem as the restriction (6). Finally, we give the adjustment problem (7) for this pair of optimization problems.

We use the following notation:

$$\begin{aligned} &K = V \setminus \{1\}, \\ &D = \{(i,j) \in V \times V : \{i,j\} \in E\}. \end{aligned} \tag{8}$$

Define for $(i,j) \in D$ nonnegative variables $x_{ij} \in \mathbb{R}$ and introduce for $k \in K$ and $(i,j) \in D$ auxiliary nonnegative variables $f_{ij}^k \in \mathbb{R}$. It is known (see e.g. [2]), that the following continous linear programming problem (9) may be used to calculate the minimum weight spanning tree T^* in graph $G = (V, E, c)$. Namely, given an optimal solution x^*, f^* of problem (9), the optimal tree T^* is composed of the edges $\{i,j\} \in E$ for which $x_{ij}^* + x_{ji}^* > 0$.

$$\max \sum_{\{i,j\} \in E} -c_{ij} \cdot (x_{ij} + x_{ji}) \tag{9a}$$

$$x_{ij} - f_{ij}^k \geq 0, \; k \in K, (i,j) \in D \tag{9b}$$

$$\begin{cases} \sum_{(i,j)\in D} x_{ij} = n - 1 \\ \sum_{(j,1)\in D} f_{j1}^k - \sum_{(1,j)\in D} f_{1j}^k = -1, \; k \in K \\ \sum_{(j,k)\in D} f_{jk}^k - \sum_{(k,j)\in D} f_{kj}^k = 1, \; k \in K \\ \sum_{(j,i)\in D} f_{ji}^k - \sum_{(i,j)\in D} f_{ij}^k = 0, \; i,k \in K, i \neq k \end{cases} \tag{9c}$$

$$\begin{cases} x_{ij} \geq 0, \; (i,j) \in D \\ f_{ij}^k \geq 0, \; k \in K, (i,j) \in D. \end{cases} \tag{9d}$$

Parts (9a), (9b), (9c) in the above formulation of the MST problem correspond, respectively, to (4a), (4b), (4c) in the initial linear program (4).

To get the MHP problem, being a restriction of (9), it is necessary to add the following constraints:

$$\sum_{j \in V}(x_{ij} + x_{ji}) \leq 2,\ i \in V, \tag{10}$$

$$x_{ij} \in \mathbb{B},\ (i,j) \in D. \tag{11}$$

Observe that constraints (10) correspond to linear constraints (5). The obtained optimization problem corresponds to (6).

Let $w_{ij}^k \geq 0$ for $k \in K$, $(i,j) \in D$, be dual variables for inequalities (9b), and $l \in \mathbb{R}$, $v_1^k \in \mathbb{R}$ for $k \in K$, $v_k^k \in \mathbb{R}$ for $k \in K$, $v_i^k \in \mathbb{R}$ for $i,k \in K$, $i \neq k$ denote dual variables for consecutive groups of constraints in problem (9c). The dual problem of linear program (9) can be stated as follows:

$$\min \sum_{k \in K} -(v_k^k - v_1^k) - (n-1) \cdot l \tag{12a}$$

$$\begin{cases} v_j^k - v_i^k - w_{ij}^k \leq 0,\ k \in K, (i,j) \in D \\ \sum_{k \in K} w_{ij}^k + l \leq c_{ij},\ \{i,j\} \in E \\ \sum_{k \in K} w_{ji}^k + l \leq c_{ij},\ \{i,j\} \in E \\ w_{ij}^k \geq 0,\ k \in K, (i,j) \in D \\ v_i^k \in \mathbb{R},\ i \in V, k \in K \\ l \in \mathbb{R}. \end{cases} \tag{12b}$$

We may now formulate the adjustment problem (7) for the considered pair of problems. It is enough to use constraints (12b) as constraints (7b) and constraints (9b), (9c), (9d), (10), (11) as constraints (7d). The constraint (7c) corresponds now to the objective functions (9a), (12a), and has the following form:

$$\sum_{k \in K}(v_k^k - v_1^k) + (n-1) \cdot l - \sum_{\{i,j\} \in E} c_{ij}(x_{ij} + x_{ji}) - \sum_{(i,j) \in D} (\delta_{ij}^+ - \delta_{ij}^-) x_{ij} = 0, \tag{13}$$

where $\delta_{ij}^+, \delta_{ij}^- \geq 0$ for $(i,j) \in D$.

Finally, the adjustment problem for a pair: the MST and the MHP problems is stated as follows:

$$\min \sum_{(i,j)\in D} (\delta_{ij}^{+} - \delta_{ij}^{-}) \tag{14a}$$

$$\begin{cases} v_j^k - v_i^k - w_{ij}^k \leq 0,\ k \in K, (i,j) \in D \\ \sum_{k\in K} w_{ij}^k + l - \delta_{ij}^{+} + \delta_{ij}^{-} \leq c_{ij},\ \{i,j\} \in E \\ \sum_{k\in K} w_{ji}^k + l - \delta_{ji}^{+} + \delta_{ji}^{-} \leq c_{ij},\ \{i,j\} \in E \\ w_{ij}^k \geq 0,\ k \in K, (i,j) \in D \\ v_i^k \in \mathbb{R},\ i \in V, k \in K \\ l \in \mathbb{R} \end{cases} \tag{14b}$$

$$\sum_{k\in K}(v_k^k - v_1^k) + (n-1)\cdot l - \sum_{\{i,j\}\in E} c_{ij}(x_{ij} + x_{ji}) - \sum_{(i,j)\in D} (\delta_{ij}^{+} - \delta_{ij}^{-}) x_{ij} = 0 \tag{14c}$$

$$\begin{cases} \sum_{(i,j)\in D} x_{ij} = n-1 \\ \sum_{(j,1)\in D} f_{j1}^k - \sum_{(1,j)\in D} f_{1j}^k = -1,\ k \in K \\ \sum_{(j,k)\in D} f_{jk}^k - \sum_{(k,j)\in D} f_{kj}^k = 1,\ k \in K \\ \sum_{(j,i)\in D} f_{ji}^k - \sum_{(i,j)\in D} f_{ij}^k = 0,\ i,k \in K, i \neq k \\ x_{ij} - f_{ij}^k \geq 0,\ k \in K, (i,j) \in D \\ x_{ij} \geq 0,\ (i,j) \in D \\ f_{ij}^k \geq 0,\ k \in K, (i,j) \in D \end{cases} \tag{14d}$$

$$\begin{cases} \sum_{j\in V}(x_{ij} + x_{ji}) \leq 2,\ i \in V \\ x_{ij} \in \{0,1\},\ (i,j) \in D \\ \delta_{ij}^{+} = \delta_{ji}^{+} \geq 0,\ (i,j) \in D \\ \delta_{ij}^{-} = \delta_{ji}^{-} \geq 0,\ (i,j) \in D. \end{cases} \tag{14e}$$

To linearize terms $\delta_{ij}^{+} x_{ij}$, $\delta_{ij}^{-} x_{ij}$, $(i,j) \in D$, we use a standard technique (see e.g. [3]). Namely, we introduce new nonnegative variables z_{ij}^{+}, z_{ij}^{-}, $(i,j) \in D$, satisfying the following conditions:

$$z_{ij}^{+} = \delta_{ij}^{+} x_{ij},\ (i,j) \in D, \tag{15a}$$

$$z_{ij}^{-} = \delta_{ij}^{-} x_{ij},\ (i,j) \in D. \tag{15b}$$

Now we can replace the constraint (14c) in the problem (14), with the following linear constraint:

$$\sum_{k\in K}(v_k^k - v_1^k) + (n-1)\cdot l - \sum_{\{i,j\}\in E} c_{ij}(x_{ij}+x_{ji}) - \sum_{(i,j)\in D} z_{ij}^+ + \sum_{(i,j)\in D} z_{ij}^- = 0. \quad (16)$$

For any new variable z_{ij}^+, z_{ij}^-, $(i,j)\in D$, we have to add also constraints which would guarantee that equations (15a) and (15b) hold. Let us take for example the equation $z_{ij}^+ = \delta_{ij}^+ x_{ij}$ for some ordered pair $(i,j)\in D$. This equation is equivalent to two implications:

$$\begin{aligned} x_{ij} = 0 &\Rightarrow z_{ij}^+ = 0, \\ x_{ij} = 1 &\Rightarrow z_{ij}^+ = \delta_{ij}^+, \end{aligned} \quad (17)$$

which can be modeled by adding the following new constraints:

$$\begin{aligned} & z_{ij}^+ - Mx_{ij} \le 0, \\ & -\delta_{ij}^+ + z_{ij}^+ \le 0, \\ & \delta_{ij}^+ - z_{ij}^+ + Mx_{ij} \le M, \end{aligned} \quad (18)$$

where M is a sufficiently large constant satisfying the inequality $\delta_{ij}^+ \le M$ for any $(i,j)\in D$. It can be shown (see [1]) that for $\Delta = \mathbb{R}^n$ we can take $M = \sum_{\{i,j\}\in E} |c_{ij}|$. So, to linearize all nonlinear terms in (14c) it is necessary to add for any $(i,j)\in D$ the following inequalities:

$$\begin{aligned} & z_{ij}^+ - Mx_{ij} \le 0 \\ & -\delta_{ij}^+ + z_{ij}^+ \le 0 \\ & \delta_{ij}^+ - z_{ij}^+ + Mx_{ij} \le M \\ & z_{ij}^- - Mx_{ij} \le 0 \\ & -\delta_{ij}^- + z_{ij}^- \le 0 \\ & -\delta_{ij}^- - z_{ij}^- + Mx_{ij} \le M \\ & z_{ij}^+, z_{ij}^- \ge 0. \end{aligned} \quad (19)$$

3 Computational results

In this section we describe computational experiment provided to test our MIP approach for the adjustment problem.

We used randomly generated graphs with the number of vertices $|V| = 11, \ldots, 20$. Following families of benchmarks were considered:

1. *complete1* – complete graphs with the number of vertices $|V| = 11, \ldots, 20$, in which the weight of each edge is an integer random variable uniformly distributed in the interval $[0; |V|^2]$. For each number of vertices 10 instances were generated.

2. *complete2* – complete graphs with the number of vertices $|V| = 11, \ldots, 20$, in which the weight of each edge is taken with the same probability from the set $\{0, 1\}$. For each number of vertices 10 instances were generated.

3. *density* – graphs with 20 vertices in which each edge appears with the probability $d = 0.2, 0.3, \ldots, 1.0$. The weight of an edge is an integer random variable uniformly distributed in the interval $[0; |V|^2]$. For each density considered 10 instances were generated.

4. *data* – a family of three groups of benchmarks (*integer*, *boolean*, *real*), constructed to investigate the influence of data (discrete, continous). In the *integer* and *real* group the weight of each edge is an integer or a continous random variable uniformly distributed in the interval $[0; |V|^2]$, respectively. In the *boolean* group the weight of each edge is taken with the same probability from the set $\{0, 1\}$. For each group considered 10 instances of graphs with $|V| = 20$ vertices were generated.

A computer program in Java was used to generate all benchmarks and related MIP problems. The appendix contains a listing of this program. To solve these MIP problems we use in our program ILOG CPLEX 10 solver libraries. For all instances optimal solutions for the adjustment problem were obtained. Resulting MIP problems for $|V| = 20$ had roughly 17000 variables (including 380 binaries), 17500 constraints, and 66500 nonzeros.

Computational times (in seconds) for AMD Opteron Processor 248 with 4 GB RAM for families *complete1*, *complete2*, *density* and *data* are presented in the following tables and figures. For any 10 instances from these families, we computed the following statistic parameters:

1. *min* – minimum computational time,

2. *median* – median of computational times,

3. *avg* – average of computational times,

4. *stddev* – standard deviation of computational times,

5. *max* – maximum computational time.

complete1 benchmarks	Number of vertices 11	12	13	14	15	16	17	18	19	20
min	0.15	0.19	0.27	0.42	1.1	1.59	2.09	2.99	7.29	10.37
median	0.19	1.24	2.47	1.62	3.92	2.63	31.23	9.49	13.8	27.2
avg	0.38	2.35	3.78	6.17	10.78	25.53	87.64	42.77	101.41	252.49
stddev	0.59	2.67	3.6	7.69	17.66	35.44	148.78	54.35	198.91	516.4
max	2.07	7.42	9.95	22.39	58.52	103.08	492.37	158.64	644.21	1552.35

Table 1: Computational times (in seconds) for the *complete1* family.

complete2 benchmarks	Number of vertices 11	12	13	14	15	16	17	18	19	20
min	0.11	0.3	0.56	0.8	1.82	2.69	5.96	9.08	11.89	16.94
median	0.19	0.75	1.55	2.02	3.45	14.83	33.74	83.57	115.31	300.77
avg	0.19	0.67	1.57	2.61	6.56	23.73	37.27	88.67	192.18	332.26
stddev	0.06	0.28	0.86	2.16	5.68	28.43	19.01	54.84	265.18	309.52
max	0.33	1.1	2.92	6.77	18.15	95.04	75.12	205.92	794.86	1042.16

Table 2: Computational times (in seconds) for the *complete2* family.

density benchmarks	Graphs density 0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
min	0.6	0.51	0.91	1.5	1.96	2.94	3.63	13.68	10.37
median	4.29	4.81	13.58	15.34	106.31	26.65	82.61	327.24	27.2
avg	4.34	8.66	29.06	38.03	164.58	46.43	215.98	683.23	252.49
stddev	2.96	9.31	37.33	47.62	173.37	54.04	275.54	734.15	516.4
max	9.47	27.95	98.11	139.64	493.52	182.67	830.16	1627.78	1552.35

Table 3: Computational times (in seconds) for the *density* family.

data benchmarks	Data type *boolean*	*integer*	*real*
min	16.94	10.37	9.19
median	300.77	27.2	124.14
avg	332.26	252.49	534.27
stddev	309.52	516.4	764.78
max	1042.16	1552.35	1980.44

Table 4: Computational times (in seconds) for the *data* family.

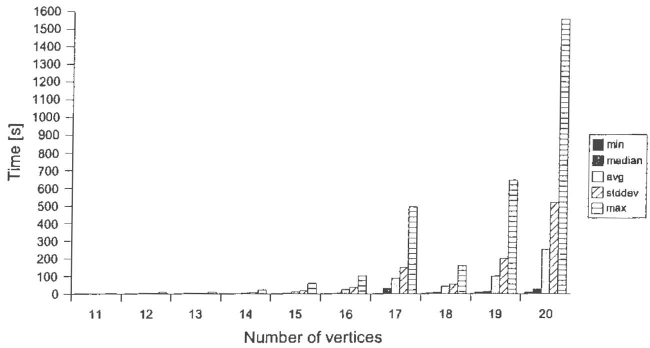

Figure 1: Computational times (in seconds) for the *complete1* family.

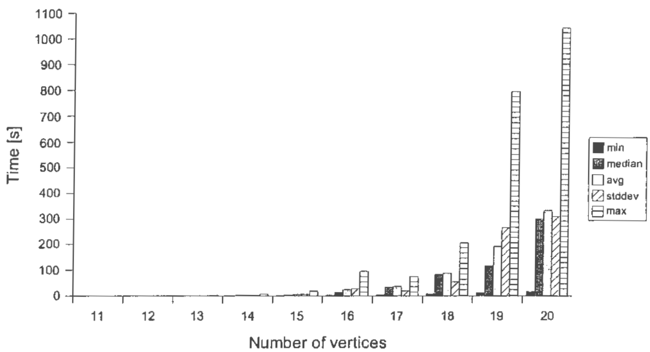

Figure 2: Computational times (in seconds) for the *complete2* family.

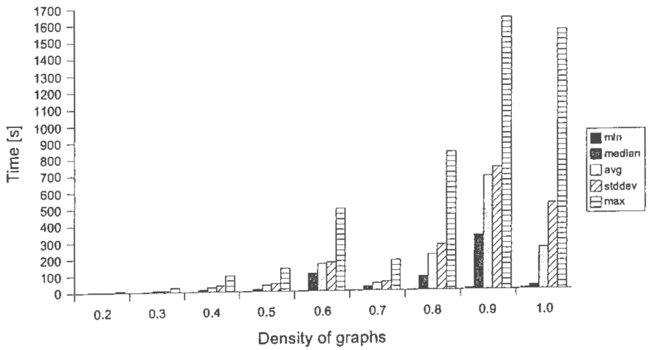

Figure 3: Computational times (in seconds) for the *density* family.

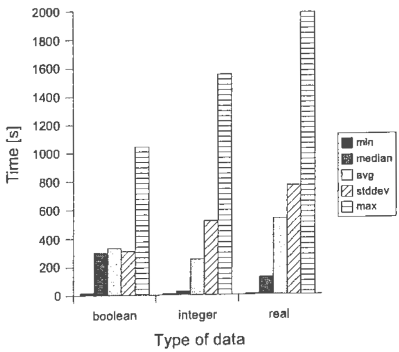

Figure 4: Computational times (in seconds) for the *data* family.

In Table 1 and Figure 1, Table 2 and Figure 2, Table 3 and Figure 3 computational results for families *complete1*, *complete2*, *density* are summarized, respectively. One can observe very fast grow of the computational time as a function of the number of vertices (*complete1* and *complete2* families) and the density (*density* family) of graphs. Only for benchmarks with 18 vertices for the *complete1* family, 17 vertices for the *complete2* family, 0.7 and 1.0 densities for the *density* family, a decrease of the computational time appeared; probably it is related to changes in ILOG CPLEX optimization strategies, but this problem requires further investigations.

In Table 4 and Figure 4 computational results for *data* family of three groups of benchmarks (*integer*, *boolean*, *real*) are summarized. The relations of the computational times for these groups are rather complicated. The *max* and *stddev* values are the smallest for problems from *boolean* group, and the largest for *real* group, but the *median* value appeared the smallest for *integer* group, and the largest for *boolean* group.

Our computational results were rather limited, for example only standard ILOG CPLEX optimization strategies were used, but all the results indicate that the MIP formulation of the adjustment problem for the considered pair of problems (MST and MHP) leads to difficult optimization problems and is limited to graphs with small number of vertices.

References

[1] M. Libura. On the adjustment problem for linear programs. *European Journal of Operational Research* 183 (2007) 125-134.

[2] T. L. Magnanti, L. A. Wolsey. Optimal trees. In: M. O. Ball et. al. (Eds.), Network Models, Handbook in Operations Research and Management Science, 7. North-Holland, Amsterdam 1995.

[3] H. P. Williams. Model Building in Mathematical Programming. John Wiley & Sons: Chichester; 1993.

4 The appendix

This section contains a listing of Java program used to generate all benchmarks and related MIP problems presented in this paper.

```
import java.util.*;
import java.io.*;
import ilog.cplex.*;
import ilog.concert.*;


public class Global {
    final static int MaxGraphSize = 1000;
    public final static double inf = Double.POSITIVE_INFINITY;
    static Random randomizer = new Random(
        java.lang.System.currentTimeMillis());

    public Global() {
    }

    public static void initRandomizer(long seed) {
        randomizer = new Random(seed);
    }
}


public class Edge {
    protected int x = -1;
    protected int y = -1;
    protected double value = 0.0;

    public Edge(int newX, int newY) {
        setEdge(newX, newY);
    }

    protected void setEdge(int newX, int newY) {
        if(newX < newY) {
            x = newX;
            y = newY;
        }
        else {
            x = newY;
            y = newX;
        }
    }

    public Edge(int newX, int newY, double newValue) {
        setEdge(newX, newY);
        value = newValue;
```

```
    }

    public double getValue() {
        return value;
    }

    public void setValue(double newValue) {
        value = newValue;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int hashCode() {
        return x * Global.MaxGraphSize + y;
    }

    public String toString() {
        return "{" + x + "," + y + "}=" + value;
    }
}


public class Graph {
    protected LinkedHashMap<Integer, Edge> edges;
    protected LinkedHashSet<Integer> vertices;
    public int liczIteracje = 0;
    public int licz = 0;

    public Graph() {
        clear();
    }

    public int verticesNumber() {
        return vertices.size();
    }

    public int edgesNumber() {
        return edges.size();
    }

    public Graph(int size) throws Exception {
        if(size < 1 || size > Global.MaxGraphSize) {
            throw new Exception("graph size out of range");
```

```
}
    edges = new LinkedHashMap<Integer, Edge>();
    vertices = new LinkedHashSet<Integer>();
    for(int i=0; i<size; i++) {
      addVertex(i);
    }
  }

  public Graph(int newSize, double density, double lbound, double
    ubound, boolean isInteger) throws Exception {
    randomGraph(newSize, density, lbound, ubound, isInteger);
  }

  public void randomGraph(int size, double density, double lbound,
    double ubound, boolean isInteger) throws Exception {
    clear();
if(density < 0.0 || density > 1.0) {
      throw new Exception("graph density out of range");
}
if(size < 1 || size > Global.MaxGraphSize) {
      throw new Exception("graph size out of range");
}
    ArrayList<Edge> edgeList = new ArrayList<Edge>();
for(int i=0; i<size; i++) {
      addVertex(i);
      for(int j=i+1; j<size; j++) {
        double value = Global.randomizer.nextDouble() *
          (ubound - lbound) + lbound;
        if(isInteger) {
          value = (double) Math.round(value);
        }
        edgeList.add(new Edge(i, j, value));
      }
    }
    for(int i=(int)(density*edgeList.size()); i>0; i--) {
      int edgeNumber = (int) (edgeList.size() *
        Global.randomizer.nextDouble());
      addEdge(edgeList.get(edgeNumber));
      edgeList.remove(edgeNumber);
    }
    sortEdgesLexOrder();
  }

  public void sortEdgesLexOrder() {
    Integer[] edgeHashesArray =
      edges.keySet().toArray(new Integer[1]);
    Arrays.sort(edgeHashesArray);
    LinkedHashMap<Integer, Edge> newEdges =
      new LinkedHashMap<Integer, Edge>();
```

```
        for(int i=0; i<edgeHashesArray.length; i++) {
            newEdges.put(edgeHashesArray[i],
                edges.get(edgeHashesArray[i]));
        }
        edges = newEdges;
    }

    public void sortVerticesLexOrder() {
        Integer[] verticesArray =
            vertices.toArray(new Integer[1]);
        Arrays.sort(verticesArray);
        LinkedHashSet<Integer> newVertices =
            new LinkedHashSet<Integer>();
        for(int i=0; i<verticesArray.length; i++) {
            newVertices.add(verticesArray[i]);
        }
        vertices = newVertices;
    }

    public LinkedHashMap<Double, LinkedHashSet<Edge>>
        sortEdgesByValues() {
        sortEdgesLexOrder();
     LinkedHashMap<Double, LinkedHashSet<Edge>>
            edgesClassifiedByValues =
            new LinkedHashMap<Double, LinkedHashSet<Edge>>();
Iterator<Edge> edgesIter = edgesIterator();
while(edgesIter.hasNext()) {
            Edge edge = edgesIter.next();
            double value = edge.getValue();
            if(!edgesClassifiedByValues.keySet().contains(value)) {
edgesClassifiedByValues.put(value, new LinkedHashSet<Edge>());
            }
            edgesClassifiedByValues.get(value).add(edge);
        }
Double[] sortedValues =
            edgesClassifiedByValues.keySet().toArray(new Double[1]);
LinkedHashMap<Double, LinkedHashSet<Edge>> edgesClassifiedByValuesSorted
            = new LinkedHashMap<Double, LinkedHashSet<Edge>>();
Arrays.sort(sortedValues);
for(int i=0; i<sortedValues.length; i++) {
            double value = sortedValues[i];
            edgesClassifiedByValuesSorted.put(value,
                edgesClassifiedByValues.get(value));
}
        LinkedHashMap<Integer, Edge> edgesSortedByValues =
            new LinkedHashMap<Integer, Edge>();
        for(int i=0; i<sortedValues.length; i++) {
            Iterator<Edge> edgesCategoryIter =
                edgesClassifiedByValuesSorted.get(
```

```
                  sortedValues[i]).iterator();
            while(edgesCategoryIter.hasNext()) {
                Edge edge = edgesCategoryIter.next();
                edgesSortedByValues.put(edge.hashCode(), edge);
            }
        }
        edges = edgesSortedByValues;
        return edgesClassifiedByValuesSorted;
    }

    public Iterator<Edge> edgesIterator() {
        return edges.values().iterator();
    }

    public Iterator<Integer> verticesIterator() {
        return vertices.iterator();
    }

    public boolean equals(Object obj) {
        if(obj.getClass() != Graph.class) return false;
        Graph that = (Graph) obj;
        if(vertices.size()!=that.vertices.size()) return false;
        Iterator<Integer> verticesIter = vertices.iterator();
        while(verticesIter.hasNext()) {
            int vertex = verticesIter.next();
            if(!that.vertices.contains(vertex)) return false;
        }
        if(edges.size()!=that.edges.size()) return false;
        Iterator<Edge> edgesIter = edgesIterator();
        while(edgesIter.hasNext()) {
            Edge edge = edgesIter.next();
            if(!that.edges.containsKey(edge.hashCode())) return false;
        }
        return true;
    }

    public boolean isClique() {
        return edgesNumber()==(verticesNumber()*(verticesNumber()-1))/2;
    }

    public void addAllEdges(Graph g) throws Exception {
        Iterator<Edge> gIter = g.edgesIterator();
        while(gIter.hasNext()) {
            Edge edge = gIter.next();
            addEdge(edge);
        }
    }

    public void addEdge(Edge edge) throws Exception {
```

```
        if(containsEdge(edge)) {
            throw new Exception("Edge " + edge.toString() +
                " is already in graph");
        }
        addVertex(edge.getX());
        addVertex(edge.getY());
        edges.put(edge.hashCode(), edge);
    }

    public void addVertex(Integer v) {
        vertices.add(v);
    }

    public void addEdge(int x, int y) throws Exception {
        addEdge(x, y, 0.0);
    }

    public void addEdge(int x, int y, double value) throws Exception {
        Edge edge = new Edge(x, y, value);
        addEdge(edge);
    }

    public void removeEdge(int x, int y) {
        Edge edge = new Edge(x, y);
        edges.remove(edge.hashCode());
    }

    public void removeAllEdges(Graph g) {
        Iterator<Edge> gIter = g.edgesIterator();
        while(gIter.hasNext()) {
            Edge edge = gIter.next();
            this.removeEdge(edge.getX(), edge.getY());
        }
    }

    public double getEdgeValue(Edge e) throws Exception {
        if(!edges.keySet().contains(e.hashCode())) {
            throw new Exception("Edge not in graph");
        }
        return edges.get(e.hashCode()).getValue();
    }

    public double getEdgeValue(int x, int y) throws Exception {
        return getEdgeValue(new Edge(x,y));
    }

    public double getGraphValue() {
        double value = 0.0;
        Iterator<Edge> graphIter = edgesIterator();
```

```
        while(graphIter.hasNext()) {
            Edge edge = graphIter.next();
            value += edge.getValue();
        }
        return value;
    }

    public void clear() {
        this.edges = new LinkedHashMap<Integer, Edge>();
        this.vertices = new LinkedHashSet<Integer>();
    }

    public boolean containsEdge(Edge edge) {
        return edges.containsKey(edge.hashCode());
    }

    public boolean containsEdge(int x, int y) {
        Edge edge = new Edge(x, y);
        return containsEdge(edge);
    }

    public String toString() {
        String string = "[V={";
        Iterator<Integer> verticesIter = vertices.iterator();
        while(verticesIter.hasNext()) {
            Integer vertex = verticesIter.next();
            string = string.concat(vertex.toString());
            if(verticesIter.hasNext()) {
                string = string.concat(", ");
            }
        }
        string = string.concat("}, E={");
        Iterator<Edge> graphIter = this.edgesIterator();
        while(graphIter.hasNext()) {
            Edge edge = graphIter.next();
            string = string.concat(edge.toString());
            if(graphIter.hasNext()) {
                string = string.concat(", ");
            }
        }
        string = string.concat("}]=" + getGraphValue());
        return string;
    }

    public Graph copy() throws Exception {
        Graph newGraph = new Graph();
        Iterator<Integer> verticesIter = vertices.iterator();
        while(verticesIter.hasNext()) {
            int vertex = verticesIter.next();
```

```
            newGraph.addVertex(vertex);
        }
        Iterator<Edge> edgesIter = edgesIterator();
        while(edgesIter.hasNext()) {
            Edge edge = edgesIter.next();
            newGraph.addEdge(edge);
        }
        return newGraph;
    }

    public boolean isTreeHamilton() {
        int[] stopnie = new int[verticesNumber()];
Iterator<Edge> drzewoIter = edgesIterator();
while(drzewoIter.hasNext()) {
            Edge para = drzewoIter.next();
            stopnie[para.getX()]++;
            stopnie[para.getY()]++;
}
boolean hamiltonPath = true;
for(int i=0; i<verticesNumber(); i++) {
            if(stopnie[i]>2) {
hamiltonPath = false;
                break;
            }
}
return hamiltonPath;
    }

    public static void main(String[] args) {
        try {
            Graph graph = new Graph(5, 0.9, 1.0, 1.0, false);
            System.out.println(graph);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}


class Solution {
    public String status;
    public Double value;
    public Double time;
}


public class Adjustment {
    public Adjustment() {
```

```
    }

    static Graph graph = new Graph();
    static String varname = "";
    static LinkedHashMap<String, String> options;
    static LinkedHashMap<String, LinkedHashSet<Solution>>
        statsSolutionStatus;
    static Random randomizer = null;

    static void initStatsSolutionStatus() {
        statsSolutionStatus = new LinkedHashMap<String,
            LinkedHashSet<Solution>>();
statsSolutionStatus.put("Bounded", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Error", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Bounded", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Feasible", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Infeasible", new LinkedHashSet<Solution>());
statsSolutionStatus.put("InfeasibleOrUnbounded",
            new LinkedHashSet<Solution>());
statsSolutionStatus.put("Optimal", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Unbounded", new LinkedHashSet<Solution>());
statsSolutionStatus.put("Unknown", new LinkedHashSet<Solution>());
    }

    static void createSampleGraph() throws Exception {
        graph.addEdge(0, 1, 4.0);
        graph.addEdge(0, 2, 1.0);
        graph.addEdge(0, 3, 4.0);
        graph.addEdge(1, 3, 5.0);
        graph.addEdge(2, 3, 3.0);
        graph.addEdge(2, 4, 7.0);
        graph.addEdge(3, 4, 8.0);
    }

    static void randGraph(int newSize, double density) throws Exception {
        double uBound = Double.parseDouble(options.get("ubound"));
        double lBound = Double.parseDouble(options.get("lbound"));
        boolean isInteger = false;
        if(options.get("valtype").compareTo("integer")==0) {
            isInteger = true;
        }
        graph.randomGraph(newSize, density, lBound, uBound, isInteger);
    }

    static void readGraph(String path) {
        try {
            File file = new File(path);
            if(!file.exists()) {
                throw new Exception("File \"" + path +
```

```
                    "\" does not exist");
            }
            BufferedReader stream =
                new BufferedReader(new FileReader(path));
            String row;
            row = stream.readLine();
            int size = Integer.parseInt(row);
            graph = new Graph();
            int rowcnt = 0;
            while((row = stream.readLine()) != null) {
                String[] numbers = row.split(" ");
if(numbers.length < size) {
                    throw new Exception("File error");
}
for(int i=0; i<size; i++) {
                    double value = Double.parseDouble(numbers[i]);
                    if(!graph.containsEdge(rowcnt,i) &&
                        value<Global.inf) {
                        graph.addEdge(rowcnt,i,value);
                    }
}
rowcnt++;
            }
            stream.close();
            graph.sortVerticesLexOrder();
            graph.sortEdgesLexOrder();
}
catch(Exception e) {
            e.printStackTrace();
}
    }

    static void saveGraph(String path) {
        try {
            File file = new File(path);
            BufferedWriter stream =
                new BufferedWriter(new FileWriter(path));
            int size = graph.verticesNumber();
            stream.write(Integer.toString(size) + "\n");
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    if(graph.containsEdge(i,j)) {
                        stream.write(Double.toString(
                            graph.getEdgeValue(i,j)) + " ");
                    }
                    else {
                        stream.write(Double.toString(Global.inf) + " ");
                    }
                }
```

```
                stream.write("\n");
            }
            stream.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    static void printGraph() {
        System.out.println(graph.toString());
    }

    static void populateByRow(IloMPModeler model,
        LinkedHashMap<String, IloNumVar> variables, ArrayList<IloRange>
        range, boolean adjustment) throws Exception {
        double lb = 0.0;
        double ub = 0.0;
        IloNumVarType type;
        IloLinearNumExpr obj = model.linearNumExpr();
double M = 0.0;
        int size = graph.verticesNumber();
        for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                if(graph.containsEdge(i,j)) {
                    M += Math.abs(graph.getEdgeValue(i,j));
                }
            }
        }
        for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
         if(graph.containsEdge(i,j)) {
                    type = IloNumVarType.Bool;
                    lb = 0.0;
                    ub = 1.0;
                    varname = "x_" + i + "_" + j;
                    variables.put(varname, model.numVar(lb, ub,
                        type, varname));
                    if(!adjustment) {
obj.addTerm(variables.get(varname),
                            graph.getEdgeValue(i,j));
                    }
}
            }
}
        for(int k=1; k<size; k++) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    if(graph.containsEdge(i,j)) {
```

```
lb = 0.0;
ub = Double.MAX_VALUE;
type = IloNumVarType.Float;
varname = "f_" + i + "_" + j + "_" + k;
variables.put(varname, model.numVar(lb, ub, type, varname));
                    }
                }
            }
        }
for(int k=0; k<2; k++) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    if(graph.containsEdge(i,j)) {
type = IloNumVarType.Float;
lb = 0.0;
ub = Double.MAX_VALUE;
if(k==0) varname = "u_";
                        if(k==1) varname = "l_";
varname += i + "_" + j;
variables.put(varname, model.numVar(lb, ub, type, varname));
if(adjustment) {
                            obj.addTerm(variables.get(varname), 1.0);
}
                    }
                }
            }
        }
for(int k=0; k<2; k++) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    if(graph.containsEdge(i,j)) {
type = IloNumVarType.Float;
lb = 0.0;
ub = Double.MAX_VALUE;
if(k==0) varname = "zu_";
if(k==1) varname = "zl_";
varname += i + "_" + j;
variables.put(varname, model.numVar(lb, ub, type, varname));
                    }
}
            }
        }
        for(int i=0; i<size; i++) {
            for(int k=1; k<size; k++) {
                type = IloNumVarType.Float;
                lb = -Double.MAX_VALUE;
ub = Double.MAX_VALUE;
varname = "v_" + i + "_" + k;
variables.put(varname, model.numVar(lb, ub, type, varname));
```

```
            }
}
        for(int k=1; k<size; k++) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    if(graph.containsEdge(i,j)) {
                        lb = 0.0;
ub = Double.MAX_VALUE;
                type = IloNumVarType.Float;
varname = "w_" + i + "_" + j + "_" + k;
variables.put(varname,model.numVar(lb,ub,type,varname));
                    }
                }
            }
        }
type = IloNumVarType.Float;
        lb = -Double.MAX_VALUE;
        ub = Double.MAX_VALUE;
varname = "l";
variables.put(varname, model.numVar(lb, ub, type, varname));
model.addMinimize(obj);
        int rngcount = 0;
        IloNumExpr expr = model.linearNumExpr();
        for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                if(graph.containsEdge(i,j)) {
                    expr = model.sum(expr,
                        variables.get("x_" + i + "_" + j));
                }
            }
        }
range.add(model.addEq(expr, size-1, "c" + rngcount));
rngcount++;
        for(int k=1; k<size; k++) {
            expr = model.linearNumExpr();
            for(int j=0; j<size; j++) {
                if(graph.containsEdge(j,0)) {
                    expr = model.sum(expr,
                        variables.get("f_" + j + "_" + 0 + "_" + k));
}
            }
            for(int j=0; j<size; j++) {
if(graph.containsEdge(0,j)) {
                    expr = model.diff(expr,
                        variables.get("f_" + 0 + "_" + j + "_" + k));
}
            }
            range.add(model.addEq(expr, -1.0, "c" + rngcount));
            rngcount++;
```

```
        }
        for(int k=1; k<size; k++) {
            expr = model.linearNumExpr();
            for(int j=0; j<size; j++) {
if(graph.containsEdge(j,k)) {
                    expr = model.sum(expr,
                        variables.get("f_" + j + "_" + k + "_" + k));
            }
if(graph.containsEdge(k,j)) {
                    expr = model.diff(expr,
                        variables.get("f_" + k + "_" + j + "_" + k));
}
            }
            range.add(model.addEq(expr, 1.0, "c" + rngcount));
            rngcount++;
        }
        for(int k=1; k<size; k++) {
            for(int i=1; i<size; i++) {
                if(i==k) {
                    continue;
                }
                else {
                    expr = model.linearNumExpr();
                    for(int j=0; j<size; j++) {
                        if(graph.containsEdge(i,j)) {
                            expr = model.sum(expr, model.diff(
                                variables.get("f_" + j + "_" +
                                i + "_" + k), variables.get(
                                "f_" + i + "_" + j + "_" + k)));
}
                    }
                    range.add(model.addEq(expr, 0.0, "c" + rngcount));
                    rngcount++;
                }
            }
        }
        for(int k=1; k<size; k++) {
            for(int i=0; i<size; i++) {
for(int j=0; j<size; j++) {
                    expr = model.linearNumExpr();
                    if(graph.containsEdge(i,j)) {
expr = model.diff(variables.get("x_"+i+"_"+j),
                        variables.get("f_" + i + "_" + j + "_" + k));
                range.add(model.addGe(expr,0.0,"c"+rngcount));
rngcount++;
                    }
                }
            }
        }
```

```
for(int i=0; i<size; i++) {
            expr = model.linearNumExpr();
            for(int j=0; j<size; j++) {
                if(graph.containsEdge(i,j)) {
                    expr = model.sum(expr, model.sum(variables.get(
                    "x_"+i+"_"+j),variables.get("x_"+j+"_"+i)));
}
            }
            range.add(model.addLe(expr, 2.0, "c" + rngcount));
            rngcount++;
}
for(int k=1; k<size; k++) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++) {
                    expr = model.linearNumExpr();
                    if(graph.containsEdge(i,j)) {
                        expr = model.sum(expr, model.diff(
                            model.diff(variables.get("v_"+j+"_"+k),
                            variables.get("v_"+i+"_"+k)),
                            variables.get("w_"+i+"_"+j+"_"+k)));
range.add(model.addLe(expr,0,"c"+rngcount));
rngcount++;
                    }
}
            }
        }
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                expr = model.linearNumExpr();
double rh = 0.0;
if(i<j && graph.containsEdge(i,j)) {
                    for(int k=1; k<size; k++) {
                        expr = model.sum(expr, variables.get("w_"
                            + i + "_" + j + "_" + k));
                    }
                    expr = model.sum(expr, variables.get("l"));
                    expr = model.diff(expr, variables.get("u_"
                        + i + "_" + j));
                    expr = model.sum(expr, variables.get("l_"
                        + i + "_" + j));
                    rh = graph.getEdgeValue(i,j);
   range.add(model.addLe(expr, rh, "c" + rngcount));
                    rngcount++;
                }
            }
}
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                expr = model.linearNumExpr();
```

```
double rh = 0.0;
if(i<j && graph.containsEdge(i,j)) {
                    for(int k=1; k<size; k++) {
                        expr = model.sum(expr, variables.get("w_"
                            + j + "_" + i + "_" + k));
                    }
                    expr = model.sum(expr, variables.get("l"));
                    expr = model.diff(expr, variables.get("u_"
                        + j + "_" + i));
                    expr = model.sum(expr, variables.get("l_"
                        + j + "_" + i));
                    rh = graph.getEdgeValue(j,i);
    range.add(model.addLe(expr,rh,"c"+rngcount));
                    rngcount++;
}
            }
        }
expr = model.linearNumExpr();
for(int k=1; k<size; k++) {
            expr = model.sum(expr, model.diff(variables.get("v_"
                +k+"_"+k), variables.get("v_"+0+"_"+k)));
}
expr = model.sum(expr, model.prod(variables.get("l"),size-1));
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
double c = 0.0;
if(graph.containsEdge(i,j)) {
                    c = graph.getEdgeValue(i,j);
                    expr = model.diff(expr, model.prod(variables.get(
                        "zu_" + i + "_" + j), 1.0));
                    expr = model.sum(expr, model.prod(variables.get(
                        "zl_" + i + "_" + j), 1.0));
                    expr = model.diff(expr, model.prod(c,
                        variables.get("x_" + i + "_" + j)));
}
            }
}
        range.add(model.addEq(expr, 0, "c" + rngcount));
rngcount++;
        for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                expr = model.linearNumExpr();
if(graph.containsEdge(i,j)) {
                    expr = model.diff(variables.get("zu_" + i +
                        "_" + j), model.prod(variables.get(
                        "x_" + i + "_" + j), M));
    range.add(model.addLe(expr, 0.0, "c" + rngcount));
                    rngcount++;
}
```

```
            }
        }
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
                expr = model.linearNumExpr();
if(graph.containsEdge(i,j)) {
                    expr = model.diff(variables.get("zu_" + i +
                        "_" + j), variables.get("u_" + i + "_" + j));
    range.add(model.addLe(expr, 0.0, "c" + rngcount));
                    rngcount++;
}
            }
        }
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
expr = model.linearNumExpr();
if(graph.containsEdge(i,j)) {
                    expr = model.sum(model.diff(variables.get("u_"
                        +i+"_"+j),variables.get("zu_"+i+"_"+j)),
                        model.prod(variables.get("x_"+i+"_"+j),M));
                    range.add(model.addLe(expr, M, "c" + rngcount));
                    rngcount++;
}
            }
        }
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
expr = model.linearNumExpr();
if(graph.containsEdge(i,j)) {
                    expr = model.diff(variables.get("zl_"+i+"_"+j),
                        model.prod(variables.get("x_"+i+"_"+j),M));
            range.add(model.addLe(expr,0.0,"c"+rngcount));
                    rngcount++;
}
            }
        }
for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
expr = model.linearNumExpr();
                if(graph.containsEdge(i,j)) {
                    expr = model.diff(variables.get("zl_"+i+"_"+j),
                        variables.get("l_" + i + "_" + j));
                    range.add(model.addLe(expr,0.0,"c"+rngcount));
                    rngcount++;
                }
            }
        }
        for(int i=0; i<size; i++) {
            for(int j=0; j<size; j++) {
```

```
                expr = model.linearNumExpr();
                if(graph.containsEdge(i,j)) {
                    expr = model.sum(model.diff(variables.get("l_"
                        +i+"_"+j), variables.get("zl_"+i+"_"+j)),
                        model.prod(variables.get("x_"+i+"_"+j), M));
                    range.add(model.addLe(expr, M, "c" + rngcount));
                    rngcount++;
                }
            }
        }
    }

    public static void readTest(String path) {
        try {
            File plik = new File(path);
            if(!plik.exists()) {
                throw new Exception("File \"" + path + "\" does
                    not exist");
            }
            BufferedReader strumien = new BufferedReader(new
                FileReader(path));
            String wiersz;
            options = new LinkedHashMap<String, String>();
            while((wiersz = strumien.readLine()) != null) {
                String[] entry = wiersz.split("\\s=\\s");
                if(entry.length != 2) {
                    throw new Exception("Error in: " + wiersz);
                }
                options.put(entry[0], entry[1]);
            }
            strumien.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static Solution solve(IloCplex cplex, LinkedHashMap<String,
        IloNumVar> variables, ArrayList<IloRange> range)
        throws Exception {
        IloRange[] rng = new IloRange[range.size()];
        for(int i=0; i<range.size(); i++) {
            rng[i] = range.get(i);
        }
        IloNumVar[] var = new IloNumVar[variables.size()];
        String[] varnames = new String[variables.size()];
        Iterator<IloNumVar> iter = variables.values().iterator();
        int i=0;
        while(iter.hasNext()) {
```

```
                IloNumVar numvar = iter.next();
                var[i] = numvar;
                varnames[i] = numvar.getName();
                i++;
            }
            if(options.get("solve").compareTo("true")==0) {
                Solution solution = new Solution();
                long timerStart = java.lang.System.currentTimeMillis();
                if(cplex.solve()) {
                    double[] x = cplex.getValues(var);
                    int ncols = cplex.getNcols();
                    for (int j = 0; j < ncols; ++j) {
                        if(x[j]!=0.0 && j<var.length) {
                            cplex.output().println("" + varnames[j] +
                                "\t=\t" + x[j]);
                        }
                    }
                }
                solution.status = cplex.getStatus().toString();
                solution.time = (double)
                    (java.lang.System.currentTimeMillis()
                    - timerStart) / 1000.0;
                if(cplex.getStatus() == IloCplex.Status.Feasible ||
                    cplex.getStatus() == IloCplex.Status.Optimal) {
                    solution.value = cplex.getObjValue();
                }
                statsSolutionStatus.get(cplex.getStatus().toString())
                    .add(solution);
                return solution;
            }
            return null;
        }

        public static void main(String[] args) {
            try {
                if(args.length != 1) {
                    throw new Exception("Incorrect number of arguments");
                }
                initStatsSolutionStatus();
                readTest(args[0]);
                long seed = 0;
                int maxTests = Integer.parseInt(options.get("testcount"));
                if(options.get("source").compareTo("random")==0) {
                    if(options.get("seed").compareTo("time")==0) {
                        seed = java.lang.System.currentTimeMillis();
}
else {
                        seed = Integer.parseInt(options.get("seed"));
}
```

```
}
Global.initRandomizer(seed);
BufferedWriter stream = null;
if(options.get("name").compareTo("none")!=0) {
    stream = new BufferedWriter(new
        FileWriter(options.get("name") + ".log"));
    stream.write("" + maxTests + "\n\n");
}
for(int countTests=0; countTests<maxTests; countTests++) {
    if(options.get("source").compareTo("random")==0) {
        randGraph(Integer.parseInt(options.get("size")),
            Double.parseDouble(options.get("density")));
    }
    if(options.get("source").compareTo("file")==0) {
        readGraph(options.get("inpath"));
    }
    printGraph();
    LinkedHashMap<String, IloNumVar> variables = new
        LinkedHashMap<String, IloNumVar>();
    ArrayList<IloRange> range = new ArrayList<IloRange>();
    IloCplex cplex = new IloCplex();
    populateByRow(cplex, variables, range, true);
    Solution solution = solve(cplex, variables, range);
    if(options.get("name").compareTo("none")!=0) {
        String path = "";
        path += options.get("name")+"_"+countTests+".lp";
        cplex.exportModel(path);
        stream.write("number\t= " + countTests + "\n");
        stream.write("status\t= " + cplex.getStatus() + "\n");
        stream.write("value\t= " + solution.value + "\n");
        stream.write("time\t= " + solution.time + "\n");
        stream.write("\n");
        stream.flush();
        saveGraph(options.get("name")+"_"+countTests
            +".graph");
    }
    cplex.end();
}
if(stream!=null) {
    Iterator<Solution> iter =
        statsSolutionStatus.get("Optimal").iterator();
    double optimalSolutions =
        statsSolutionStatus.get("Optimal").size();
    double timeavg = 0.0;
    double timemax = 0.0;
    double timemin = 999999999.0;
    while(iter.hasNext()) {
        Solution solution = iter.next();
        timeavg += solution.time / optimalSolutions;
```

```
                        if(timemax < solution.time) {
                            timemax = solution.time;
                        }
                        if(timemin > solution.time) {
                            timemin = solution.time;
                        }
                    }
                    stream.write("\n");
                    stream.write("optimal\t= "+(int)optimalSolutions+"\n");
                    stream.write("timemin\t= " + timemin + "\n");
                    stream.write("timeavg\t= " + timeavg + "\n");
                    stream.write("timemax\t= " + timemax + "\n");
                    stream.close();
                }
            }
catch(IloCplex.UnknownObjectException iloException) {
                System.out.println(iloException.getObject().toString());
}
            catch(Exception e) {
                e.printStackTrace();
            }
        }
}
```