

69/2007

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 hab. 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

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

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:

$$\max\{c^T x : x \in F\}. \quad (2)$$

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:

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

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:

$$\max c^T x \quad (4a)$$

$$A'x \leq b' \quad (4b)$$

$$A''x = b'' \quad (4c)$$

$$x \geq 0, \quad (4d)$$

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

$$Cx \leq d, \quad (5)$$

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 $V = \{1, \dots, n\}$, $E \subseteq \{\{i, j\} : i, j \in V\}$, $c \in \mathbb{R}^{|E|}$. 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:

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

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 continuous 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 $x_{ij}^* + x_{ji}^* > 0$.

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

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

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

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

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:

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

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

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:

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

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

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:

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

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:

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

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

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

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

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

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:

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

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

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

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

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:

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

where M is a sufficiently large constant satisfying the inequality $\delta_{ij}^+ \leq M$ 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:

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

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 $|V| = 11, \dots, 20$. Following families of benchmarks were considered:

1. *complete1* – complete graphs with the number of vertices $|V| = 11, \dots, 20$, 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 $|V| = 11, \dots, 20$, 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 $d = 0.2, 0.3, \dots, 1.0$. 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, continuous). In the *integer* and *real* group the weight of each edge is an integer or a continuous 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.

<i>complete1</i> benchmarks	Number of vertices									
	11	12	13	14	15	16	17	18	19	20
<i>min</i>	0.15	0.19	0.27	0.42	1.1	1.59	2.09	2.99	7.29	10.37
<i>median</i>	0.19	1.24	2.47	1.62	3.92	2.63	31.23	9.49	13.8	27.2
<i>avg</i>	0.38	2.35	3.78	6.17	10.78	25.53	87.64	42.77	101.41	252.49
<i>stddev</i>	0.59	2.67	3.6	7.69	17.66	35.44	148.78	54.35	198.91	516.4
<i>max</i>	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.

<i>complete2</i> benchmarks	Number of vertices									
	11	12	13	14	15	16	17	18	19	20
<i>min</i>	0.11	0.3	0.56	0.8	1.82	2.69	5.96	9.08	11.89	16.94
<i>median</i>	0.19	0.75	1.55	2.02	3.45	14.83	33.74	83.57	115.31	300.77
<i>avg</i>	0.19	0.67	1.57	2.61	6.56	23.73	37.27	88.67	192.18	332.26
<i>stddev</i>	0.06	0.28	0.86	2.16	5.68	28.43	19.01	54.84	265.18	309.52
<i>max</i>	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.

<i>density</i> benchmarks	Graphs density									
	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
<i>min</i>	0.6	0.51	0.91	1.5	1.96	2.94	3.63	13.68	10.37	
<i>median</i>	4.29	4.81	13.58	15.34	106.31	26.65	82.61	327.24	27.2	
<i>avg</i>	4.34	8.66	29.06	38.03	164.58	46.43	215.98	683.23	252.49	
<i>stddev</i>	2.96	9.31	37.33	47.62	173.37	54.04	275.54	734.15	516.4	
<i>max</i>	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.

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

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

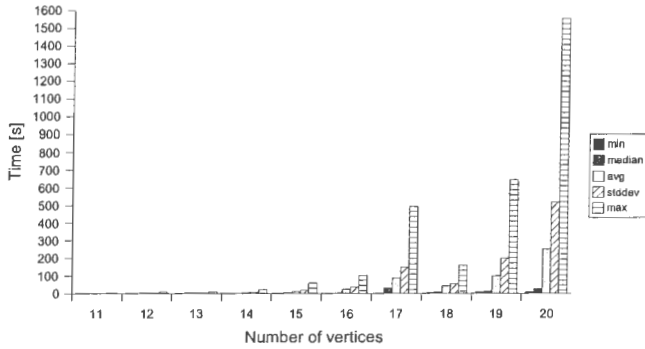


Figure 1: Computational times (in seconds) for the *complete1* family.

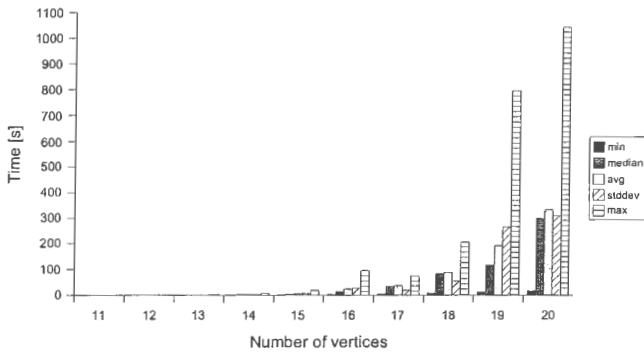


Figure 2: Computational times (in seconds) for the *complete2* family.

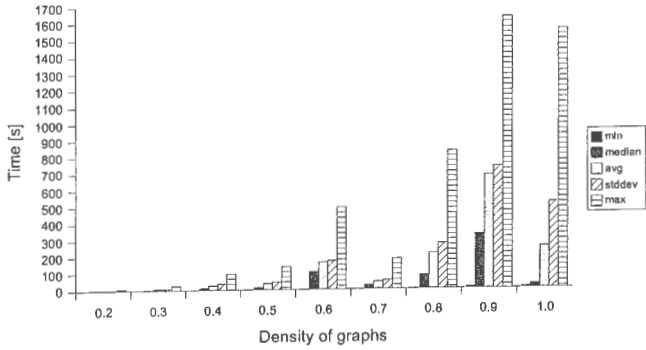


Figure 3: Computational times (in seconds) for the *density* family.

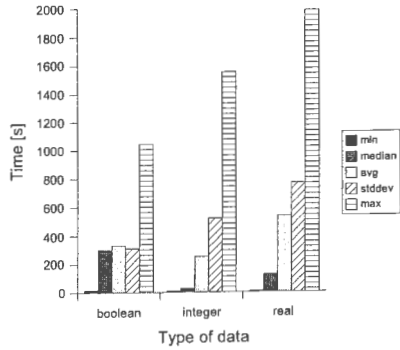


Figure 4: Computational times (in seconds) for the *data* family.

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.

```
1 import java.util.*;
2 import java.io.*;
3 import ilog.cplex.*;
4 import ilog.concert.*;
5
6
7 public class Global {
8     final static int MaxGraphSize = 1000;
9     public final static double inf = Double.POSITIVE_INFINITY;
10    static Random randomizer = new Random(
11        java.lang.System.currentTimeMillis());
12
13    public Global() {
14    }
15
16    public static void initRandomizer(long seed) {
17        randomizer = new Random(seed);
18    }
19 }
20
21
22 public class Edge {
23     protected int x = -1;
24     protected int y = -1;
25     protected double value = 0.0;
26
27     public Edge(int newX, int newY) {
28         setEdge(newX, newY);
29     }
30
31     protected void setEdge(int newX, int newY) {
32         if(newX < newY) {
33             x = newX;
34             y = newY;
35         }
36         else {
37             x = newY;
38             y = newX;
39         }
40     }
41
42     public Edge(int newX, int newY, double newValue) {
43         setEdge(newX, newY);
44         value = newValue;
45     }
46 }
```

```

45     }
46
47     public double getValue() {
48         return value;
49     }
50
51     public void setValue(double newValue) {
52         value = newValue;
53     }
54
55     public int getX() {
56         return x;
57     }
58
59     public int getY() {
60         return y;
61     }
62
63     public int hashCode() {
64         return x * Global.MaxGraphSize + y;
65     }
66
67     public String toString() {
68         return "{" + x + ", " + y + "}=" + value;
69     }
70 }
71
72
73 public class Graph {
74     protected LinkedHashMap<Integer, Edge> edges;
75     protected LinkedHashSet<Integer> vertices;
76     public int liczIteracje = 0;
77     public int licz = 0;
78
79     public Graph() {
80         clear();
81     }
82
83     public int verticesNumber() {
84         return vertices.size();
85     }
86
87     public int edgesNumber() {
88         return edges.size();
89     }
90
91     public Graph(int size) throws Exception {
92         if(size < 1 || size > Global.MaxGraphSize) {
93             throw new Exception("graph size out of range");

```



```

94     }
95     edges = new LinkedHashMap<Integer, Edge>();
96     vertices = new LinkedHashSet<Integer>();
97     for(int i=0; i<size; i++) {
98         addVertex(i);
99     }
100 }
101
102 public Graph(int newSize, double density, double lbound, double
103 ubound, boolean isInteger) throws Exception {
104     randomGraph(newSize, density, lbound, ubound, isInteger);
105 }
106
107 public void randomGraph(int size, double density, double lbound,
108 double ubound, boolean isInteger) throws Exception {
109     clear();
110     if(density < 0.0 || density > 1.0) {
111         throw new Exception("graph density out of range");
112     }
113     if(size < 1 || size > Global.MaxGraphSize) {
114         throw new Exception("graph size out of range");
115     }
116     ArrayList<Edge> edgeList = new ArrayList<Edge>();
117     for(int i=0; i<size; i++) {
118         addVertex(i);
119         for(int j=i+1; j<size; j++) {
120             double value = Global.randomizer.nextDouble() *
121                 (ubound - lbound) + lbound;
122             if(isInteger) {
123                 value = (double) Math.round(value);
124             }
125             edgeList.add(new Edge(i, j, value));
126         }
127     }
128     for(int i=(int)(density*edgeList.size()); i>0; i--) {
129         int edgeNumber = (int) (edgeList.size() *
130             Global.randomizer.nextDouble());
131         addEdge(edgeList.get(edgeNumber));
132         edgeList.remove(edgeNumber);
133     }
134     sortEdgesLexOrder();
135 }
136
137 public void sortEdgesLexOrder() {
138     Integer[] edgeHashesArray =
139         edges.keySet().toArray(new Integer[1]);
140     Arrays.sort(edgeHashesArray);
141     LinkedHashMap<Integer, Edge> newEdges =
142         new LinkedHashMap<Integer, Edge>();

```

```

143         for(int i=0; i<edgeHashesArray.length; i++) {
144             newEdges.put(edgeHashesArray[i],
145                 edges.get(edgeHashesArray[i]));
146         }
147         edges = newEdges;
148     }
149
150     public void sortVerticesLexOrder() {
151         Integer[] verticesArray =
152             vertices.toArray(new Integer[1]);
153         Arrays.sort(verticesArray);
154         LinkedHashSet<Integer> newVertices =
155             new LinkedHashSet<Integer>();
156         for(int i=0; i<verticesArray.length; i++) {
157             newVertices.add(verticesArray[i]);
158         }
159         vertices = newVertices;
160     }
161
162     public LinkedHashMap<Double, LinkedHashSet<Edge>>
163         sortEdgesByValues() {
164         sortEdgesLexOrder();
165         LinkedHashMap<Double, LinkedHashSet<Edge>>
166             edgesClassifiedByValues =
167             new LinkedHashMap<Double, LinkedHashSet<Edge>>();
168         Iterator<Edge> edgesIter = edgesIterator();
169         while(edgesIter.hasNext()) {
170             Edge edge = edgesIter.next();
171             double value = edge.getValue();
172             if(!edgesClassifiedByValues.keySet().contains(value)) {
173                 edgesClassifiedByValues.put(value, new LinkedHashSet<Edge>());
174             }
175             edgesClassifiedByValues.get(value).add(edge);
176         }
177         Double[] sortedValues =
178             edgesClassifiedByValues.keySet().toArray(new Double[1]);
179         LinkedHashMap<Double, LinkedHashSet<Edge>> edgesClassifiedByValuesSorted
180             = new LinkedHashMap<Double, LinkedHashSet<Edge>>();
181         Arrays.sort(sortedValues);
182         for(int i=0; i<sortedValues.length; i++) {
183             double value = sortedValues[i];
184             edgesClassifiedByValuesSorted.put(value,
185                 edgesClassifiedByValues.get(value));
186         }
187         LinkedHashMap<Integer, Edge> edgesSortedByValues =
188             new LinkedHashMap<Integer, Edge>();
189         for(int i=0; i<sortedValues.length; i++) {
190             Iterator<Edge> edgesCategoryIter =
191                 edgesClassifiedByValuesSorted.get(

```

```

192         sortedValues[i]).iterator();
193         while(edgesCategoryIter.hasNext()) {
194             Edge edge = edgesCategoryIter.next();
195             edgesSortedByValues.put(edge.hashCode(), edge);
196         }
197     }
198     edges = edgesSortedByValues;
199     return edgesClassifiedByValuesSorted;
200 }
201
202 public Iterator<Edge> edgesIterator() {
203     return edges.values().iterator();
204 }
205
206 public Iterator<Integer> verticesIterator() {
207     return vertices.iterator();
208 }
209
210 public boolean equals(Object obj) {
211     if(obj.getClass() != Graph.class) return false;
212     Graph that = (Graph) obj;
213     if(vertices.size() != that.vertices.size()) return false;
214     Iterator<Integer> verticesIter = vertices.iterator();
215     while(verticesIter.hasNext()) {
216         int vertex = verticesIter.next();
217         if(!that.vertices.contains(vertex)) return false;
218     }
219     if(edges.size() != that.edges.size()) return false;
220     Iterator<Edge> edgesIter = edgesIterator();
221     while(edgesIter.hasNext()) {
222         Edge edge = edgesIter.next();
223         if(!that.edges.containsKey(edge.hashCode())) return false;
224     }
225     return true;
226 }
227
228 public boolean isClique() {
229     return edgesNumber() == (verticesNumber() * (verticesNumber() - 1)) / 2;
230 }
231
232 public void addAllEdges(Graph g) throws Exception {
233     Iterator<Edge> gIter = g.edgesIterator();
234     while(gIter.hasNext()) {
235         Edge edge = gIter.next();
236         addEdge(edge);
237     }
238 }
239
240 public void addEdge(Edge edge) throws Exception {

```

```

241         if(containsEdge(edge)) {
242             throw new Exception("Edge " + edge.toString() +
243                 " is already in graph");
244         }
245         addVertex(edge.getX());
246         addVertex(edge.getY());
247         edges.put(edge.hashCode(), edge);
248     }
249
250     public void addVertex(Integer v) {
251         vertices.add(v);
252     }
253
254     public void addEdge(int x, int y) throws Exception {
255         addEdge(x, y, 0.0);
256     }
257
258     public void addEdge(int x, int y, double value) throws Exception {
259         Edge edge = new Edge(x, y, value);
260         addEdge(edge);
261     }
262
263     public void removeEdge(int x, int y) {
264         Edge edge = new Edge(x, y);
265         edges.remove(edge.hashCode());
266     }
267
268     public void removeAllEdges(Graph g) {
269         Iterator<Edge> gIter = g.edgesIterator();
270         while(gIter.hasNext()) {
271             Edge edge = gIter.next();
272             this.removeEdge(edge.getX(), edge.getY());
273         }
274     }
275
276     public double getEdgeValue(Edge e) throws Exception {
277         if(!edges.keySet().contains(e.hashCode())) {
278             throw new Exception("Edge not in graph");
279         }
280         return edges.get(e.hashCode()).getValue();
281     }
282
283     public double getEdgeValue(int x, int y) throws Exception {
284         return getEdgeValue(new Edge(x,y));
285     }
286
287     public double getGraphValue() {
288         double value = 0.0;
289         Iterator<Edge> graphIter = edgesIterator();

```

```

290         while(graphIter.hasNext()) {
291             Edge edge = graphIter.next();
292             value += edge.getValue();
293         }
294         return value;
295     }
296
297     public void clear() {
298         this.edges = new LinkedHashMap<Integer, Edge>();
299         this.vertices = new LinkedHashSet<Integer>();
300     }
301
302     public boolean containsEdge(Edge edge) {
303         return edges.containsKey(edge.hashCode());
304     }
305
306     public boolean containsEdge(int x, int y) {
307         Edge edge = new Edge(x, y);
308         return containsEdge(edge);
309     }
310
311     public String toString() {
312         String string = "[V={";
313         Iterator<Integer> verticesIter = vertices.iterator();
314         while(verticesIter.hasNext()) {
315             Integer vertex = verticesIter.next();
316             string = string.concat(vertex.toString());
317             if(verticesIter.hasNext()) {
318                 string = string.concat(", ");
319             }
320         }
321         string = string.concat("}, E={");
322         Iterator<Edge> graphIter = this.edges.iterator();
323         while(graphIter.hasNext()) {
324             Edge edge = graphIter.next();
325             string = string.concat(edge.toString());
326             if(graphIter.hasNext()) {
327                 string = string.concat(", ");
328             }
329         }
330         string = string.concat("}]=" + getGraphValue());
331         return string;
332     }
333
334     public Graph copy() throws Exception {
335         Graph newGraph = new Graph();
336         Iterator<Integer> verticesIter = vertices.iterator();
337         while(verticesIter.hasNext()) {
338             int vertex = verticesIter.next();

```

```

339         newGraph.addVertex(vertex);
340     }
341     Iterator<Edge> edgesIter = edgesIterator();
342     while(edgesIter.hasNext()) {
343         Edge edge = edgesIter.next();
344         newGraph.addEdge(edge);
345     }
346     return newGraph;
347 }
348
349     public boolean isTreeHamilton() {
350         int[] stopnie = new int[verticesNumber()];
351     Iterator<Edge> drzewoIter = edgesIterator();
352     while(drzewoIter.hasNext()) {
353         Edge para = drzewoIter.next();
354         stopnie[para.getX()]++;
355         stopnie[para.getY()]++;
356     }
357     boolean hamiltonPath = true;
358     for(int i=0; i<verticesNumber(); i++) {
359         if(stopnie[i]>2) {
360             hamiltonPath = false;
361             break;
362         }
363     }
364     return hamiltonPath;
365 }
366
367     public static void main(String[] args) {
368         try {
369             Graph graph = new Graph(5, 0.9, 1.0, 1.0, false);
370             System.out.println(graph);
371         }
372         catch(Exception e) {
373             e.printStackTrace();
374         }
375     }
376 }
377
378
379     class Solution {
380         public String status;
381         public Double value;
382         public Double time;
383     }
384
385
386     public class Adjustment {
387         public Adjustment() {

```

```

388     }
389
390     static Graph graph = new Graph();
391     static String varname = "";
392     static LinkedHashMap<String, String> options;
393     static LinkedHashMap<String, LinkedHashSet<Solution>>
394         statsSolutionStatus;
395     static Random randomizer = null;
396
397     static void initStateSolutionStatus() {
398         statsSolutionStatus = new LinkedHashMap<String,
399             LinkedHashSet<Solution>>();
400     statsSolutionStatus.put("Bounded", new LinkedHashSet<Solution>());
401     statsSolutionStatus.put("Error", new LinkedHashSet<Solution>());
402     statsSolutionStatus.put("Bounded", new LinkedHashSet<Solution>());
403     statsSolutionStatus.put("Feasible", new LinkedHashSet<Solution>());
404     statsSolutionStatus.put("Infeasible", new LinkedHashSet<Solution>());
405     statsSolutionStatus.put("InfeasibleOrUnbounded",
406         new LinkedHashSet<Solution>());
407     statsSolutionStatus.put("Optimal", new LinkedHashSet<Solution>());
408     statsSolutionStatus.put("Unbounded", new LinkedHashSet<Solution>());
409     statsSolutionStatus.put("Unknown", new LinkedHashSet<Solution>());
410     }
411
412     static void createSampleGraph() throws Exception {
413         graph.addEdge(0, 1, 4.0);
414         graph.addEdge(0, 2, 1.0);
415         graph.addEdge(0, 3, 4.0);
416         graph.addEdge(1, 3, 5.0);
417         graph.addEdge(2, 3, 3.0);
418         graph.addEdge(2, 4, 7.0);
419         graph.addEdge(3, 4, 8.0);
420     }
421
422     static void randGraph(int newSize, double density) throws Exception {
423         double uBound = Double.parseDouble(options.get("ubound"));
424         double lBound = Double.parseDouble(options.get("lbound"));
425         boolean isInteger = false;
426         if(options.get("valtype").compareTo("integer")==0) {
427             isInteger = true;
428         }
429         graph.randomGraph(newSize, density, lBound, uBound, isInteger);
430     }
431
432     static void readGraph(String path) {
433         try {
434             File file = new File(path);
435             if(!file.exists()) {
436                 throw new Exception("File \" + path +

```

```

437         "\"" does not exist");
438     }
439     BufferedReader stream =
440         new BufferedReader(new FileReader(path));
441     String row;
442     row = stream.readLine();
443     int size = Integer.parseInt(row);
444     graph = new Graph();
445     int rowcnt = 0;
446     while((row = stream.readLine()) != null) {
447         String[] numbers = row.split(" ");
448         if(numbers.length < size) {
449             throw new Exception("File error");
450         }
451         for(int i=0; i<size; i++) {
452             double value = Double.parseDouble(numbers[i]);
453             if(!graph.containsEdge(rowcnt,i) &&
454                 value<Global.inf) {
455                 graph.addEdge(rowcnt,i,value);
456             }
457         }
458         rowcnt++;
459     }
460     stream.close();
461     graph.sortVerticesLexOrder();
462     graph.sortEdgesLexOrder();
463 }
464 catch(Exception e) {
465     e.printStackTrace();
466 }
467 }
468
469     static void saveGraph(String path) {
470         try {
471             File file = new File(path);
472             BufferedWriter stream =
473                 new BufferedWriter(new FileWriter(path));
474             int size = graph.verticesNumber();
475             stream.write(Integer.toString(size) + "\n");
476             for(int i=0; i<size; i++) {
477                 for(int j=0; j<size; j++) {
478                     if(graph.containsEdge(i,j)) {
479                         stream.write(Double.toString(
480                             graph.getEdgeValue(i,j)) + " ");
481                     }
482                     else {
483                         stream.write(Double.toString(Global.inf) + " ");
484                     }
485                 }

```



```

486         stream.write("\n");
487     }
488     stream.close();
489 }
490 catch(Exception e) {
491     e.printStackTrace();
492 }
493 }
494
495 static void printGraph() {
496     System.out.println(graph.toString());
497 }
498
499 static void populateByRow(IloMPModeler model,
500     LinkedHashMap<String, IloNumVar> variables, ArrayList<IloRange>
501     range, boolean adjustment) throws Exception {
502     double lb = 0.0;
503     double ub = 0.0;
504     IloNumVarType type;
505     IloLinearNumExpr obj = model.linearNumExpr();
506     double M = 0.0;
507     int size = graph.verticesNumber();
508     for(int i=0; i<size; i++) {
509         for(int j=0; j<size; j++) {
510             if(graph.containsEdge(i,j)) {
511                 M += Math.abs(graph.getEdgeValue(i,j));
512             }
513         }
514     }
515     for(int i=0; i<size; i++) {
516         for(int j=0; j<size; j++) {
517             if(graph.containsEdge(i,j)) {
518                 type = IloNumVarType.Bool;
519                 lb = 0.0;
520                 ub = 1.0;
521                 varname = "x_" + i + "_" + j;
522                 variables.put(varname, model.numVar(lb, ub,
523                     type, varname));
524                 if(!adjustment) {
525                     obj.addTerm(variables.get(varname),
526                         graph.getEdgeValue(i,j));
527                 }
528             }
529         }
530     }
531     for(int k=1; k<size; k++) {
532         for(int i=0; i<size; i++) {
533             for(int j=0; j<size; j++) {
534                 if(graph.containsEdge(i,j)) {

```

```

535 lb = 0.0;
536 ub = Double.MAX_VALUE;
537 type = IloNumVarType.Float;
538 varname = "f_" + i + "_" + j + "_" + k;
539 variables.put(varname, model.numVar(lb, ub, type, varname));
540     }
541     }
542     }
543 }
544 for(int k=0; k<2; k++) {
545     for(int i=0; i<size; i++) {
546         for(int j=0; j<size; j++) {
547             if(graph.containsEdge(i,j)) {
548                 type = IloNumVarType.Float;
549                 lb = 0.0;
550                 ub = Double.MAX_VALUE;
551                 if(k==0) varname = "u_";
552                 if(k==1) varname = "l_";
553                 varname += i + "_" + j;
554                 variables.put(varname, model.numVar(lb, ub, type, varname));
555                 if(adjustment) {
556                     obj.addTerm(variables.get(varname), 1.0);
557                 }
558             }
559         }
560     }
561 }
562 for(int k=0; k<2; k++) {
563     for(int i=0; i<size; i++) {
564         for(int j=0; j<size; j++) {
565             if(graph.containsEdge(i,j)) {
566                 type = IloNumVarType.Float;
567                 lb = 0.0;
568                 ub = Double.MAX_VALUE;
569                 if(k==0) varname = "zu_";
570                 if(k==1) varname = "zl_";
571                 varname += i + "_" + j;
572                 variables.put(varname, model.numVar(lb, ub, type, varname));
573             }
574         }
575     }
576 }
577     for(int i=0; i<size; i++) {
578         for(int k=1; k<size; k++) {
579             type = IloNumVarType.Float;
580             lb = -Double.MAX_VALUE;
581             ub = Double.MAX_VALUE;
582             varname = "v_" + i + "_" + k;
583             variables.put(varname, model.numVar(lb, ub, type, varname));

```

```

584     }
585 }
586     for(int k=1; k<size; k++) {
587         for(int i=0; i<size; i++) {
588             for(int j=0; j<size; j++) {
589                 if(graph.containsEdge(i,j)) {
590                     lb = 0.0;
591 ub = Double.MAX_VALUE;
592                 type = IloNumVarType.Float;
593 varname = "w_" + i + "_" + j + "_" + k;
594 variables.put(varname, model.numVar(lb,ub,type,varname));
595                 }
596             }
597         }
598     }
599 type = IloNumVarType.Float;
600 lb = -Double.MAX_VALUE;
601 ub = Double.MAX_VALUE;
602 varname = "l";
603 variables.put(varname, model.numVar(lb, ub, type, varname));
604 model.addMinimize(obj);
605     int rngcount = 0;
606     IloNumExpr expr = model.linearNumExpr();
607     for(int i=0; i<size; i++) {
608         for(int j=0; j<size; j++) {
609             if(graph.containsEdge(i,j)) {
610                 expr = model.sum(expr,
611                     variables.get("x_" + i + "_" + j));
612             }
613         }
614     }
615 range.add(model.addEq(expr, size-1, "c" + rngcount));
616 rngcount++;
617     for(int k=1; k<size; k++) {
618         expr = model.linearNumExpr();
619         for(int j=0; j<size; j++) {
620             if(graph.containsEdge(j,0)) {
621                 expr = model.sum(expr,
622                     variables.get("f_" + j + "_" + 0 + "_" + k));
623             }
624         }
625         for(int j=0; j<size; j++) {
626             if(graph.containsEdge(0,j)) {
627                 expr = model.diff(expr,
628                     variables.get("f_" + 0 + "_" + j + "_" + k));
629             }
630         }
631         range.add(model.addEq(expr, -1.0, "c" + rngcount));
632         rngcount++;

```

```

633     }
634     for(int k=1; k<size; k++) {
635         expr = model.linearNumExpr();
636         for(int j=0; j<size; j++) {
637             if(graph.containsEdge(j,k)) {
638                 expr = model.sum(expr,
639                     variables.get("f_" + j + "_" + k + "_" + k));
640             }
641             if(graph.containsEdge(k,j)) {
642                 expr = model.diff(expr,
643                     variables.get("f_" + k + "_" + j + "_" + k));
644             }
645         }
646         range.add(model.addEq(expr, 1.0, "c" + rngcount));
647         rngcount++;
648     }
649     for(int k=1; k<size; k++) {
650         for(int i=1; i<size; i++) {
651             if(i==k) {
652                 continue;
653             }
654             else {
655                 expr = model.linearNumExpr();
656                 for(int j=0; j<size; j++) {
657                     if(graph.containsEdge(i,j)) {
658                         expr = model.sum(expr, model.diff(
659                             variables.get("f_" + j + "_" +
660                                 i + "_" + k), variables.get(
661                                 "f_" + i + "_" + j + "_" + k));
662                     }
663                 }
664                 range.add(model.addEq(expr, 0.0, "c" + rngcount));
665                 rngcount++;
666             }
667         }
668     }
669     for(int k=1; k<size; k++) {
670         for(int i=0; i<size; i++) {
671             for(int j=0; j<size; j++) {
672                 expr = model.linearNumExpr();
673                 if(graph.containsEdge(i,j)) {
674                     expr = model.diff(variables.get("x_" + i + "_" + j),
675                         variables.get("f_" + i + "_" + j + "_" + k));
676                     range.add(model.addGe(expr, 0.0, "c" + rngcount));
677                 }
678             }
679         }
680     }
681 }

```

```

682 for(int i=0; i<size; i++) {
683     expr = model.linearNumExpr();
684     for(int j=0; j<size; j++) {
685         if(graph.containsEdge(i,j)) {
686             expr = model.sum(expr, model.sum(variables.get(
687                 "x_"+i+"_"+j),variables.get("x_"+j+"_"+i)));
688         }
689     }
690     range.add(model.addLe(expr, 2.0, "c" + rngcount));
691     rngcount++;
692 }
693 for(int k=1; k<size; k++) {
694     for(int i=0; i<size; i++) {
695         for(int j=0; j<size; j++) {
696             expr = model.linearNumExpr();
697             if(graph.containsEdge(i,j)) {
698                 expr = model.sum(expr, model.diff(
699                     model.diff(variables.get("v_"+j+"_"+k),
700                         variables.get("v_"+i+"_"+k)),
701                     variables.get("w_"+i+"_"+j+"_"+k)));
702             range.add(model.addLe(expr,0,"c"+rngcount));
703             rngcount++;
704         }
705     }
706 }
707 }
708 for(int i=0; i<size; i++) {
709     for(int j=0; j<size; j++) {
710         expr = model.linearNumExpr();
711         double rh = 0.0;
712         if(i<j && graph.containsEdge(i,j)) {
713             for(int k=1; k<size; k++) {
714                 expr = model.sum(expr, variables.get("w_"
715                     + i + "_" + j + "_" + k));
716             }
717             expr = model.sum(expr, variables.get("l"));
718             expr = model.diff(expr, variables.get("u_"
719                 + i + "_" + j));
720             expr = model.sum(expr, variables.get("l_"
721                 + i + "_" + j));
722             rh = graph.getEdgeValue(i,j);
723             range.add(model.addLe(expr, rh, "c" + rngcount));
724             rngcount++;
725         }
726     }
727 }
728 for(int i=0; i<size; i++) {
729     for(int j=0; j<size; j++) {
730         expr = model.linearNumExpr();

```

```

731 double rh = 0.0;
732 if(i<j && graph.containsEdge(i,j)) {
733     for(int k=1; k<size; k++) {
734         expr = model.sum(expr, variables.get("w_"
735             + j + "_" + i + "_" + k));
736     }
737     expr = model.sum(expr, variables.get("l"));
738     expr = model.diff(expr, variables.get("u_"
739         + j + "_" + i));
740     expr = model.sum(expr, variables.get("l_"
741         + j + "_" + i));
742     rh = graph.getEdgeValue(j,i);
743     range.add(model.addLe(expr,rh,"c"+rngcount));
744     rngcount++;
745 }
746     }
747 }
748 expr = model.linearNumExpr();
749 for(int k=1; k<size; k++) {
750     expr = model.sum(expr, model.diff(variables.get("v_"
751         +k+"_" +k), variables.get("v_"+0+"_" +k)));
752 }
753 expr = model.sum(expr, model.prod(variables.get("l"),size-1));
754 for(int i=0; i<size; i++) {
755     for(int j=0; j<size; j++) {
756         double c = 0.0;
757         if(graph.containsEdge(i,j)) {
758             c = graph.getEdgeValue(i,j);
759             expr = model.diff(expr, model.prod(variables.get(
760                 "zu_" + i + "_" + j), 1.0));
761             expr = model.sum(expr, model.prod(variables.get(
762                 "zl_" + i + "_" + j), 1.0));
763             expr = model.diff(expr, model.prod(c,
764                 variables.get("x_" + i + "_" + j)));
765         }
766     }
767 }
768     range.add(model.addEq(expr, 0, "c" + rngcount));
769     rngcount++;
770     for(int i=0; i<size; i++) {
771         for(int j=0; j<size; j++) {
772             expr = model.linearNumExpr();
773             if(graph.containsEdge(i,j)) {
774                 expr = model.diff(variables.get("zu_" + i +
775                     "_" + j), model.prod(variables.get(
776                         "x_" + i + "_" + j), M));
777                 range.add(model.addLe(expr, 0.0, "c" + rngcount));
778                 rngcount++;
779             }

```

```

780     }
781   }
782   for(int i=0; i<size; i++) {
783     for(int j=0; j<size; j++) {
784       expr = model.linearNumExpr();
785       if(graph.containsEdge(i,j)) {
786         expr = model.diff(variables.get("zu_" + i +
787           "_" + j), variables.get("u_" + i + "_" + j));
788         range.add(model.addLe(expr, 0.0, "c" + rngcount));
789         rngcount++;
790       }
791     }
792   }
793   for(int i=0; i<size; i++) {
794     for(int j=0; j<size; j++) {
795       expr = model.linearNumExpr();
796       if(graph.containsEdge(i,j)) {
797         expr = model.sum(model.diff(variables.get("u_"
798           +i+"_"+j),variables.get("zu_"+"i+"_"+j)),
799           model.prod(variables.get("x_"+"i+"_"+j),M));
800         range.add(model.addLe(expr, M, "c" + rngcount));
801         rngcount++;
802       }
803     }
804   }
805   for(int i=0; i<size; i++) {
806     for(int j=0; j<size; j++) {
807       expr = model.linearNumExpr();
808       if(graph.containsEdge(i,j)) {
809         expr = model.diff(variables.get("zl_"+"i+"_"+j),
810           model.prod(variables.get("x_"+"i+"_"+j),M));
811         range.add(model.addLe(expr,0.0,"c"+rngcount));
812         rngcount++;
813       }
814     }
815   }
816   for(int i=0; i<size; i++) {
817     for(int j=0; j<size; j++) {
818       expr = model.linearNumExpr();
819       if(graph.containsEdge(i,j)) {
820         expr = model.diff(variables.get("zl_"+"i+"_"+j),
821           variables.get("l_" + i + "_" + j));
822         range.add(model.addLe(expr,0.0,"c"+rngcount));
823         rngcount++;
824       }
825     }
826   }
827   for(int i=0; i<size; i++) {
828     for(int j=0; j<size; j++) {

```

```

829         expr = model.linearNumExpr();
830         if(graph.containsEdge(i,j)) {
831             expr = model.sum(model.diff(variables.get("l_"
832                 +i+"_"+j), variables.get("z1_"+i+"_"+j)),
833                 model.prod(variables.get("x_"+i+"_"+j), M));
834             range.add(model.addLe(expr, M, "c" + rngcount));
835             rngcount++;
836         }
837     }
838 }
839 }
840
841 public static void readTest(String path) {
842     try {
843         File plik = new File(path);
844         if(!plik.exists()) {
845             throw new Exception("File \"\" + path + "\" does
846                 not exist");
847         }
848         BufferedReader strumien = new BufferedReader(new
849             FileReader(path));
850         String wiersz;
851         options = new LinkedHashMap<String, String>();
852         while((wiersz = strumien.readLine()) != null) {
853             String[] entry = wiersz.split("\\s=\\s");
854             if(entry.length != 2) {
855                 throw new Exception("Error in: " + wiersz);
856             }
857             options.put(entry[0], entry[1]);
858         }
859         strumien.close();
860     }
861     catch(Exception e) {
862         e.printStackTrace();
863     }
864 }
865
866 public static Solution solve(IloCplex cplex, LinkedHashMap<String,
867     IloNumVar> variables, ArrayList<IloRange> range)
868     throws Exception {
869     IloRange[] rng = new IloRange[range.size()];
870     for(int i=0; i<range.size(); i++) {
871         rng[i] = range.get(i);
872     }
873     IloNumVar[] var = new IloNumVar[variables.size()];
874     String[] varnames = new String[variables.size()];
875     Iterator<IloNumVar> iter = variables.values().iterator();
876     int i=0;
877     while(iter.hasNext()) {

```



```

878         IloNumVar numvar = iter.next();
879         var[i] = numvar;
880         varnames[i] = numvar.getName();
881         i++;
882     }
883     if(options.get("solve").compareTo("true")==0) {
884         Solution solution = new Solution();
885         long timerStart = java.lang.System.currentTimeMillis();
886         if(cplex.solve()) {
887             double[] x = cplex.getValues(var);
888             int ncols = cplex.getNcols();
889             for (int j = 0; j < ncols; ++j) {
890                 if(x[j]!=0.0 && j<var.length) {
891                     cplex.output().println(" + varnames[j] +
892                         "\t=\t" + x[j]);
893                 }
894             }
895         }
896         solution.status = cplex.getStatus().toString();
897         solution.time = (double)
898             (java.lang.System.currentTimeMillis()
899              - timerStart) / 1000.0;
900         if(cplex.getStatus() == IloCplex.Status.Feasible ||
901            cplex.getStatus() == IloCplex.Status.Optimal) {
902             solution.value = cplex.getObjValue();
903         }
904         statsSolutionStatus.get(cplex.getStatus().toString())
905             .add(solution);
906         return solution;
907     }
908     return null;
909 }
910
911 public static void main(String[] args) {
912     try {
913         if(args.length != 1) {
914             throw new Exception("Incorrect number of arguments");
915         }
916         initStatsSolutionStatus();
917         readTest(args[0]);
918         long seed = 0;
919         int maxTests = Integer.parseInt(options.get("testcount"));
920         if(options.get("source").compareTo("random")==0) {
921             if(options.get("seed").compareTo("time")==0) {
922                 seed = java.lang.System.currentTimeMillis();
923             }
924         } else {
925             seed = Integer.parseInt(options.get("seed"));
926         }

```

```

927     }
928     Global.initRandomizer(seed);
929     BufferedWriter stream = null;
930     if(options.get("name").compareTo("none")!=0) {
931         stream = new BufferedWriter(new
932             FileWriter(options.get("name") + ".log"));
933         stream.write("#" + maxTests + "\n\n");
934     }
935     for(int countTests=0; countTests<maxTests; countTests++) {
936         if(options.get("source").compareTo("random")==0) {
937             randGraph(Integer.parseInt(options.get("size")),
938                 Double.parseDouble(options.get("density")));
939         }
940         if(options.get("source").compareTo("file")==0) {
941             readGraph(options.get("inpath"));
942         }
943         printGraph();
944         LinkedHashMap<String, IloNumVar> variables = new
945             LinkedHashMap<String, IloNumVar>();
946         ArrayList<IloRange> range = new ArrayList<IloRange>();
947         IloCplex cplex = new IloCplex();
948         populateByRow(cplex, variables, range, true);
949         Solution solution = solve(cplex, variables, range);
950         if(options.get("name").compareTo("none")!=0) {
951             String path = "";
952             path += options.get("name")+"_"+countTests+".lp";
953             cplex.exportModel(path);
954             stream.write("number\t= " + countTests + "\n");
955             stream.write("status\t= " + cplex.getStatus() + "\n");
956             stream.write("value\t= " + solution.value + "\n");
957             stream.write("time\t= " + solution.time + "\n");
958             stream.write("\n");
959             stream.flush();
960             saveGraph(options.get("name")+"_"+countTests
961                 + ".graph");
962         }
963         cplex.end();
964     }
965     if(stream!=null) {
966         Iterator<Solution> iter =
967             statsSolutionStatus.get("Optimal").iterator();
968         double optimalSolutions =
969             statsSolutionStatus.get("Optimal").size();
970         double timeavg = 0.0;
971         double timemax = 0.0;
972         double timemin = 999999999.0;
973         while(iter.hasNext()) {
974             Solution solution = iter.next();
975             timeavg += solution.time / optimalSolutions;

```

```

976         if(timemax < solution.time) {
977             timemax = solution.time;
978         }
979         if(timemin > solution.time) {
980             timemin = solution.time;
981         }
982     }
983     stream.write("\n");
984     stream.write("optimal\t= "+(int)optimalSolutions+"\n");
985     stream.write("timemin\t= " + timemin + "\n");
986     stream.write("timeavg\t= " + timeavg + "\n");
987     stream.write("timemax\t= " + timemax + "\n");
988     stream.close();
989 }
990 }
991 catch(IloCplex.UnknownObjectException iloException) {
992     System.out.println(iloException.getObject().toString());
993 }
994     catch(Exception e) {
995         e.printStackTrace();
996     }
997 }
998 }

```