Applied Mathematics and Computation 237 (2014) 85–97

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

A mixed integer linear programming model and variable neighborhood search for Maximally Balanced Connected Partition Problem

Dragan Matić

University of Banjaluka, Faculty of Mathematics and Natural Sciences, Mladena Stojanovića 2, 78000 Banjaluka, Bosnia and Herzegovina

ARTICLE INFO

Keywords:
Balanced graphs
Graph partitioning
Mixed integer linear programming
Variable neighborhood search
Combinatorial optimization

ABSTRACT

This paper deals with Maximally Balanced Connected Partition (MBCP) problem. It introduces a mixed integer linear programming (MILP) formulation of the problem with polynomial number of variables and constraints. Also, a variable neighborhood search (VNS) technique for solving this problem is presented. The VNS implements the suitable neighborhoods based on changing the component for an increasing number of vertices. An efficient implementation of the local search procedure yields a relatively short running time. The numerical experiments are made on instances known in the literature. Based on the MILP model, tests are run using two MILP solvers, CPLEX and Gurobi. It is shown that both solvers succeed in finding optimal solutions for all smaller and most of medium scale instances. Proposed VNS reaches most of the optimal solutions. The algorithm is also successfully tested on large scale problem instances for which optimal solutions are not known.

1. Introduction

Let $G = (V, E)$, $V = \{1, 2, \ldots, n\}$ be a connected graph, $|E| = m$ and let w_i be weights on vertices. For any subset $V' \subset V$ the value $w(V')$ is defined as a sum of weights of all vertices belonging to V', i.e. $w(V') = \sum_{i \in V'} w_i$. The problem considered in this paper (Maximally Balanced Connected Partition Problem – MBCP) is to find a partition (V_1, V_2) of V into nonempty disjoint sets V_1 and V_2 such that subgraphs of G induced by V_1 and V_2 are connected and the value $obj(V_1, V_2) = |w(V_1) - w(V_2)|$ is minimized.

Example 1. Let us consider the graph shown in Fig. 1. The labels of the vertices are $1, 2, \ldots, 6$, while the weights are given in brackets, near to the labels. The total sum of weights is 30. Ideally, the partition would contain two components with the equal weights (15). In that case, the vertex 5, with the weight 10, would be put in the component with the vertex 1, or with the vertices 3 and 6, but in both cases, the components $\{1, 5\}$ and $\{3, 5, 6\}$ are not connected. So, the optimal solution can not be equal to zero, but at least 2, because the total sum of all weights is even. One optimal solution is $(V_1, V_2) = (\{3, 4, 5\}, \{1, 2, 6\})$, and the $obj(V_1, V_2) = |16 - 14| = 2$. The other optimal solution is $(V_1', V_2') = (\{1, 2, 3, 6\}, \{4, 5\})$, with the equal $obj(V_1', V_2') = |16 - 14| = 2$.

E-mail address: matic.dragan@gmail.com

http://dx.doi.org/10.1016/j.amc.2014.03.098
0096-3003/

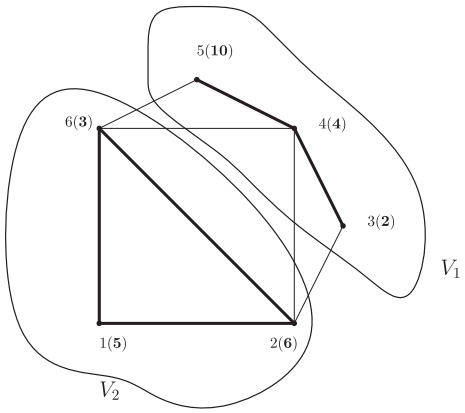

Fig. 1. The graph and the solution of the MBCP.

The first notable theoretical results in analyzing this problem are presented in [1]. In this paper, the author proved that this problem is NP hard and suggested a simple polynomial time approximation algorithm with an excellent guaranteed bound 1.072.

A successful implementation of genetic algorithm (GA) for solving this problem is presented in [2]. The authors proposed the GA using binary representation of a chromosome, indicating to which component each vertex is assigned. In the case of unconnected components (therefore incorrect solutions), penalty function is applied, directing the GA towards correct solution space. The reliability and the effectiveness of the GA is tested on grid graph and random generated instances, containing up to 300 vertices and 2000 edges.

The more general problem is so called BCPq (Balanced Connected Partition of graphs for q partitions, where $q \geqslant 2$). For a given integer number q, connected vertex weighted graph has to be partitioned into q partitions $V_1, \ldots, V_q$, so that each subgraph associated to each partition is connected and the weight of the lightest one is the highest possible, i.e., the distribution of the weights among the subgraphs should be the most homogeneous possible. In [3], 2-approximation algorithms are presented for BCPq for $q = 3$ and $q = 4$. That paper also contains a detailed proof of NP hardness of the BCPq. The authors also considered the case when q is not fixed (it is part of the instance), showing that problem does not admit an approximation algorithm with ratio smaller than 6/5, unless $P = NP$.

A similar problem to BCPq is the (l, u) partitioning, where one wishes to partition the graph G into connected components, so that the total weight of each component is at least l and at most u. Recent research in this direction is given in [4,5]. For example, in [4], the authors deal with three problems to find an (l, u)-partition of a given graph; the minimum partition problem and maximum partition problem (finding an (l, u)-partition with the minimum (respectively maximum) number of components; and the p-partition problem (finding an (l, u)-partition with a fixed number p of components). Although (l, u) partitioning is NP-hard in general case, for some special graphs it is solvable in a polynomial time. For example, in [4], the authors show that both the minimum partition problem and the maximum partition problem can be solved in time $O(u^4 n)$ and the p-partition problem can be solved in time $O(p^2 u^4 n)$ for any series–parallel graph with n vertices. Short analysis shows that determining the existence of the (l, u) partition is a case of BCPq: If the partition for BCPq problem is found, so that the objective value of the solution is less than δ , then (l, u) partition exists for the chosen $l = t\_sum/q - \delta$ and $u = t\_sum/q + \delta$, where t_sum is the total sum of all weights.

MBCP and BCPq belong to a wide class of graph partitioning problems and have a lot of direct and closely related applications in various fields of engineering, such as digital signal processing, image processing, managing electric power networks etc., social issues and education.

For instance, for controlling and routing in large scale wireless sensor networks, the network of N clusters is considered, where each cluster corresponds to one cluster head. In order to simplify network handling, the idea is to divide that large scale network into two balanced sub-networks, which can be handled and optimized independently. The network is modelled as an undirected connected graph, $G(H, A)$, where H is the set of cluster heads, $H = \{CH_i : i = 1..N\}$ and A is the set of all undirected links (CH_i, CH_j), where CH_i and CH_j are two cluster heads. The objective is to partition G into connected balanced subgraphs and this problem corresponds to MBCP. In [6], the authors adopted the approach proposed in [1] and used it to divide the network of clusters into two smaller, connected sub-networks.

In image processing, partitioning can be used in facing image degradation, when a picture is converted from one form to another. In the situation when there is no information about the degradation process, the only way for image enhancement is to increase contrast and reduce noise by suitable modifications of the grey level of pixels. To form a graph corresponding to the image, the set of vertices corresponding to grey levels is defined, and vertex weights are defined as the number of occurrences of the corresponding tone in the image. Finding the optimal grey scale transformation is formulated as the problem of partitioning the corresponding graph, such that the sum of the weights of the vertices in each component is "as constant as possible" [7].

When forming political districts, the objective is to divide the map of a country into several regions with an almost equal number of voters. If G is the corresponding dual graph of the map M, each vertex v of G represents a region, the value $w(v)$ represents the number of voters in region v. Two vertices in the graph are adjacent if corresponding regions are adjacent. So, the optimization problem is to find the partition of the graph, so that the corresponding regions have almost equal number of voters. In [8], a tabu search and adaptive memory heuristic is used to solve the problem.

BCPq is successfully used to improve the demarcation of areas for police patrolling [9]. This demarcation seeks to homogenize the number of crimes among the patrol regions balanced by crime rate. If the map of a particular region is taken as a graph, the objective is to organize police forces to act among the connected patrol regions. The method described in [9] is comprised of two phases: initial partitioning and partition refinement. In the first phase, a graph is divided in two connected sub-graphs and in the second phase heuristic algorithm swaps the vertices between the partitions with the intent of maximizing the homogeneity of the weights between them.

In education, solving MBCP can be useful for finding solutions to practical organizational problems. For example, the course material can naturally be divided into lessons, where the appropriate difficulty is assigned to each lesson. The connections between the lessons can be defined by various criteria, like conditionality, analogy or similarity. The idea is to divide the course material into two disjoint connected sections, so that the sections are of similar difficulty, as much as possible. Another example would be partitioning the group of students into two smaller groups. The "connectivity" between two students can be defined in several ways, for example, as "the ability to work together". The objective should be to divide a student group into two smaller, having in mind that groups should be balanced by student abilities.

The rest of the paper is organized as follows. In Section 2 a mixed integer linear programming formulation for MBCP is presented. In Section 3 the VNS algorithm for solving MBCP is described. Section 4 contains experimental results, based on the instances used in [2]. Section 5 is the conclusion.

2. Integer linear programming formulation

Representing discrete optimization problems as (mixed) integer linear programming (MILP) formulation can be useful to find the optimal solution. This is usually achieved by applying various optimization techniques and using some well known computer-based MILP solvers [10–12]. In this paper, CPLEX and Gurobi solvers are used to examine the theoretical result, the first MILP formulation for MBCP.

It is not hard to define linear programming formulation for the objective function and constraints related to vertex weights. On the other hand, the connectivity condition is more complex, especially if one wants a model with a polynomial number of constraints. One idea which could be useful is the network flow approach. This approach requires that the graph on which the network flow is constructed must be connected and each edge must have the orientation. Since it is not known to which component (V_1 or V_2) each vertex is assigned, the extra vertex (vertex 0) is added and that vertex is the starting point of the flow. The flow of the size $|V|$ is released from the vertex 0 and the aim is to leave the flow equal to 1 on each vertex of the graph. This idea will guarantee that all vertices will be included in the flow. So, the additional vertex resolves the problem of connectivity (vertex 0 is connected to all vertices in V). Since our graph is not directed, the orientation of the edges should be defined in an appropriate way. The orientation of the edges starting with the vertex 0 is clear, i.e. the vertex 0 is the starting one. To define the orientation of all other edges, the idea of a spanning tree is introduced. In trees, a natural order of the vertices exists and it goes from the parent vertex to its children. Constructing the spanning tree of each component would be enough to ensure the connectivity of the components (trees are connected). That could be easily done by limiting the number of vertices of the components G_1 and G_2 to $|V_1|-1$ and $|V_2|-1$, and excluding the appearance of cycles. For example, to avoid the cycles in the subgraph G_1, we could use the fact that the graph G_1 is acyclic if and only if all subgraphs induced by any set of vertices S, $|S|>3$, $S\subset G_1$ have at most $|S|-1$ edges. In this case the introduction of the network flow seems unnecessary. However, in this case, exponential number of constraints appears. So, the proposed model combines the ideas of the flow and the spanning tree: spanning tree defines the orientation of the edges, and the flow models avoids cycles; if a cycle with a flow greater than 0 exists in the network flow, the flow could be multiplied arbitrary many times and that is a clear contradiction. Also, it is obvious that two vertices (one per each component) are to be identified, as the starting vertices of the flow in each component.

Let $G=(V,E)$ be a graph. Firstly, a new vertex 0 is introduced, which is connected to each vertex of G. Formally, the set of vertices is extended by one vertex, $\overline{V}=V\cup\{0\}$. The set of edges is extended by $|V|$ edges which connect the vertex 0 and vertices in V : $\overline{E}=E\cup\partial E$, where $\partial E=\{(0,i):i\in V\}$. The total sum of all vertex weights is denoted as $w_{sum}=\sum_{v_i\in V}w_i$.

The aim is to find a partition (V_1,V_2) of V into nonempty disjoint sets V_1 and V_2 such that subgraphs G_1 and G_2 of G induced by V_1 and V_2, respectively are connected and the value $obj(V_1,V_2)=|w(V_1)-w(V_2)|$ is minimized. Let E_1 and E_2 be sets of edges in G_1 and G_2 respectively.

Since G_1 and G_2 are connected, they contain spanning trees, $T_1(V_1,E_1')$ and $T_2(V_2,E_2')$, respectively. Let two arbitrary vertices, $p\in V_1$ and $q\in V_2$ be identified. The sets $\overline{E}_1'=E_1'\cup\{(0,p)\}$ and $\overline{E}_2'=E_2'\cup\{(0,q)\}$ are defined, as well as graphs $T=(V,E_1'\cup E_2')$ and $\overline{T}=(\overline{V},\overline{E}_1'\cup\overline{E}_2')$.

To form the MILP model, the following sets of variables are introduced:

$$x_i=\begin{cases}1, & i\in V_1\\ 0, & i\in V_2\end{cases} \tag{1}$$

$$y_e = \begin{cases} 1 & e \in \overline{E}'_1 \\ 0 & \text{otherwise} \end{cases} \qquad z_e = \begin{cases} 1 & e \in \overline{E}'_2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$u_e \in [-n, n], \quad e \in \overline{E} \tag{3}$$

For each edge $e \in \overline{E}$, variable u_e denotes the flow of the edge e. The MILP formulation of MBCP is:

$$\min \ -w_{sum} + 2\sum_{i=1}^{n} x_i w_i \tag{4}$$

s.t.

$$2\sum_{i=1}^{n} x_i w_i \geqslant W \tag{5}$$

$$y_e \leqslant \frac{1}{2}x_{i_e} + \frac{1}{2}x_{j_e}, \quad e \in E \tag{6}$$

$$z_e \leqslant 1 - \frac{1}{2}x_{i_e} - \frac{1}{2}x_{j_e}, \quad e \in E \tag{7}$$

$$y_e \leqslant x_{j_e}, \quad e \in \partial E \tag{8}$$

$$z_e \leqslant 1 - x_{j_e}, \quad e \in \partial E \tag{9}$$

$$u_e \leqslant n \cdot y_e + n \cdot z_e, \quad e \in \overline{E} \tag{10}$$

$$u_e \geqslant -n \cdot y_e - n \cdot z_e, \quad e \in \overline{E} \tag{11}$$

$$\sum_{e:j_e=i} u_e - \sum_{e:i_e=i} u_e = 1, \quad i \in V \tag{12}$$

$$\sum_{e:i_e=0} u_e = n \tag{13}$$

$$\sum_{e\in E} y_e + \sum_{e\in E} z_e = n - 2 \tag{14}$$

$$\sum_{e\in \partial E} y_e + \sum_{e\in \partial E} z_e = 2 \tag{15}$$

$$x_i, y_e, z_e \in \{0, 1\}, \quad i \in V, \quad e \in \overline{E} \tag{16}$$

$$u_e \in [-n, n], \quad e \in \overline{E} \tag{17}$$

Following the approach of spanning trees, every edge in the spanning tree has the orientation, which is defined from the parent to the child. So, for each edge $e \in \overline{E}'_1 \cup \overline{E}'_2$, we can identify the starting and the ending vertex, denoted by i_e and j_e, respectively. For all other edges, not belonging to the SP, values y_e and z_e are equal to 0 and, because of (10) and (11), there is no flow on these edges. Therefore, the orientation of these edges does not play any role.

As can be seen, all three vectors y, z and u are related to all existing edges in the graph, as well as the edges between each vertex and extra vertex (vertex 0). It can easily be counted that there are $2m + 3n$ binary variables, $m + n$ real variables and $4m + 5n + 4$ constraints.

Now, obj_{MILP} can be defined as $obj_{MILP}(x, y, z, u) = -w_{sum} + 2\sum_{i=1}^{n} x_i w_i$ subject to (5)–(17).

Let us show that the solution of this MILP formulation is the solution of the MBCP.

Based on constraints given in the MILP formulation, the following Lemma guarantees the existence of exactly one vertex per component, connected to the additional vertex 0.

Lemma 1. *From constraints (6)–(15) and the objective function, it follows that in optimal solution the vertices $p \in V_1$ and $q \in V_2$, such that $u_{(0,p)} = |V_1|$, $u_{(0,q)} = |V_2|$ and $u_{(0,i)} = 0$, $i \neq p, q$ exist.*

Proof. Let us first consider an arbitrary edge $e \in \overline{E}$. The edge e can or does not have to be included in the spanning tree. In the first case, y_e (or z_e) needs to be equal to 1, otherwise 0. By the constraint (6), the value y is bounded by the righthand side of the inequalities, by the values x_{i_e} and x_{j_e} (similarly, the constraint (7) bounds the variable z). For example, if both i_e and j_e belong to V_1, both x_{i_e} and x_{j_e} at the same time are equal to 1, and in that case, the constraint (6) allows that the edge e can be included in the spanning tree. In fact, for x_{i_e} and x_{j_e}, there are four possibilities:

(i) $x_{i_e} = 1$ and $x_{j_e} = 1$: That means that both vertices belong to V_1;
(ii) $x_{i_e} = 1$ and $x_{j_e} = 0$: the vertex i belongs to V_1 and the vertex j belongs to V_2;
(iii) $x_{i_e} = 0$ and $x_{j_e} = 1$: the vertex i belongs to V_2 and the vertex j belongs to V_1;
(iv) $x_{i_e} = 0$ and $x_{j_e} = 0$: both vertices belong to V_2.

Only in the cases (i) and (iv), there exist the possibility that the edge e is included in the spanning tree and in cases (ii) and (iii), constraints (6) and (7) guarantee that e will not be included, and because of (10) and (11), u_e is equal to 0.

From (15), it directly follows that exactly two edges $(0,p)$ and $(0,q)$ exist. These edges belong to ∂E and for them $y_e = 1$ or $z_e = 1$. It will be shown that p and q have to belong to different subsets V_1 and V_2. Suppose that, without loss of generality, both p and q belong to V_1. From (13) it follows that $u_{(0,p)} + u_{(0,q)} = n$.

Let us sum the constraints from (12), when $i \in V_1$.

$$|V_1| = \sum_{i\in V_1}\left(\sum_{e:j_e=i} u_e - \sum_{e:i_e=i} u_e\right) = \sum_{e:j_e\in V_1} u_e - \sum_{e:i_e\in V_1} u_e$$

$$= \sum_{e:i_e\in V_1 \wedge j_e\in V_1} u_e + \sum_{e:i_e=0\wedge j_e\in V_1} u_e + \sum_{e:i_e\in V_2\wedge j_e\in V_1} u_e - \left(\sum_{e:i_e\in V_1\wedge j_e\in V_1} u_e + \sum_{e:i_e\in V_1\wedge j_e\in V_2} u_e\right) = \sum_{e:i_e=0\wedge j_e\in V_1} u_e = u_{(0,p)} + u_{(0,q)} = n$$

because the first and the fourth sum are annihilated, while the third and the last sum have all members equal to zero as it is shown above. So, the expression $|V_1| = n$ is received, implying that the objective function reaches the value w_{sum}, which is its maximum. However, this is opposite to the minimization process defined by the objective function (4), and this is in the contradiction with the conditions of the lemma. So, the proof of the lemma is finished. □

Theorem 1. *Partition $V^* = (V_1^*, V_2^*)$ is optimal if and only if there is an optimal solution (x, y, z, u) of (4)–(17).*

Proof. ($\Rightarrow$) Based on the partition V^*, the variables (x, y, z, u) will be constructed, such that $obj_{MILP}(x, y, z, u) \leqslant obj(V_1^*, V_2^*)$. Without loss of generality, $w(V_1^*) \geqslant w(V_2^*)$ is assumed. Let the variables x_i be defined according to (1), and y_e and z_e according to (2). It is obvious that (16) is satisfied.

Variables $u_e, e \in \overline{E}$ are defined as follows:

For $e \in \partial E$, we define

$$u_e = \begin{cases} |V_1^*|, & e_j = p \\ |V_2^*|, & e_j = q \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Since $|V_1^*| < n$ and $|V_2^*| < n$, $u_e \in [-n, n]$, for all $e \in \partial E$.

To determine the values u_e for all other edges, $e \in \overline{E} \setminus \partial E$, the fact that in a tree each vertex can be declared as a root is used and all the other vertices can be ordered, by some search algorithm. For example, one of the two standard search algorithms can be used for this purpose: depth-first search or breadth-first search. Let E_1' and E_2' be the spanning trees for V_1^* and V_2^* generated by the search algorithm. Using the ordering formed by the search algorithm, in a case when the vertex i_e is a parent of the vertex j_e, the value of the flow $u_{i_e j_e}$ is defined as the number of vertices in the subtree, rooted by the vertex j_e. In a case when the vertex j_e is a parent of the vertex i_e, the value of the flow $u_{i_e j_e}$ is defined as the minus number of vertices in the subtree, rooted by the vertex i_e. For all the other edges, belonging neither to E_1' nor E_2', $u_e = 0$ is proposed. Since that the number of vertices in each subtree is less or equal to n, then u_e belongs to $[-n, n]$, for each edge $e \in E$. Thus, (17) holds for each $e \in \overline{E}$.

The inequality $obj_{MILP}(x, y, z, u) \leqslant obj(V_1^*, V_2^*)$ is satisfied because $obj_{MILP}(x, y, z, u) = -w_{sum} + 2\sum_{i=1}^{n} x_i w_i = \sum_{i=1}^{n} w_i(2x_i - 1)$ $= \sum_{i\in V_1^*} w_i(2x_i - 1) + \sum_{i\in V_2^*} w_i(2x_i - 1) = \sum_{i\in V_1^*} w_i - \sum_{i\in V_2^*} w_i = w(V_1^*) - w(V_2^*) = obj(V^*)$.

Note that $obj_{MILP} = obj(V^*)$ is proven, which is a stronger condition than is needed. Also, (5) is proven, since $obj(V^*) \geqslant 0$.

Let us prove that all constraints (6)–(15) are satisfied.

To prove inequalities (6), two cases are considered:

(i) $y_e = 0$. The inequalities (6) are satisfied, because $x_{i_e},\ x_{j_e} \geqslant 0$ by definition.
(ii) $y_e = 1 \Rightarrow e \in E_1' \Rightarrow x_{i_e} = x_{j_e} = 1 \Rightarrow y_e \leqslant \frac{1}{2}x_{i_e} + \frac{1}{2}x_{j_e}$

To prove inequalities (8), two cases are considered again:

(i) $y_e = 0$. The inequalities (8) hold because $x_{j_e} \geqslant 0$ by definition.
(ii) $y_e = 1 \Rightarrow e \in \bar{E}'_1$. Since $e \in \partial E,\ e \in \bar{E}'_1 \cap \partial E = \{(0,p)\}$, which implies that $j_e = p \in V_1 \Rightarrow x_{j_e} = 1$.

The inequalities (7) and (9) are proven analogously as the last two steps.
To prove the inequalities (10) and (11) two cases are analyzed:

(i) $e \in \bar{E}'_1 \cup \bar{E}'_2$. Right hand sizes of the in Eqs. (10) and (11) are equal to n and $-n$, respectively. Since that $u_e \in [-n, n]$, the inequalities (10) and (11) are satisfied.
(ii) $e \notin \bar{E}'_1 \cup \bar{E}'_2$. Then u_e is equal to 0 by definition. The inequalities (10) and (11) are satisfied because right hand sizes of (10) are non-negative and right hand sizes of (10) are non-positive.

To prove the inequality (12), without loss of generality, suppose that $i \in V_1$. Let us consider all the edges from $\bar{E}'_1$, starting or ending by the vertex i. The one edge of these comes from the parent vertex, and all the others are successors. For all the other edges incident with i and not belonging to $\bar{E}'_1$, the values u_e are equal to 0 and they have no influence to (12).

$$\begin{aligned}\sum_{e:j_e=i} u_e - \sum_{e:i_e=i} u_e &= \sum_{e:j_e=i\wedge u_e>0} u_e + \sum_{e:j_e=i\wedge u_e<0} u_e - \sum_{e:i_e=i\wedge u_e>0} u_e - \sum_{e:i_e=i\wedge u_e<0} u_e \\ &= \sum_{e:j_e=i\wedge u_e>0} |u_e| + \sum_{e:j_e=i\wedge u_e<0} -|u_e| - \sum_{e:i_e=i\wedge u_e>0} |u_e| - \sum_{e:i_e=i\wedge u_e<0} -|u_e| \\ &= \sum_{e:j_e=i\wedge u_e>0} |u_e| + \sum_{e:i_e=i\wedge u_e<0} |u_e| - \left(\sum_{e:j_e=i\wedge u_e<0} |u_e| + \sum_{e:i_e=i\wedge u_e>0} |u_e| \right).\end{aligned}$$

In the first two sums, there is only one edge satisfying the conditions and that edge is coming from the parent of the vertex i to the vertex i. For that edge, $|u_e|$ is equal to the number of the vertices in the subtree rooted by i. In the last two sums, all the edges, coming from the vertex i to all its successors in spanning tree T_1, participate. The total sum of $|u_e|$ for these edges is equal to the total number of vertices of subtrees rooted by each successor of the vertex i. Since only vertex i participates in the subtree rooted by itself and not rooted by its successors, the value $\sum_{e:j_e=i\wedge u_e>0}|u_e| + \sum_{e:i_e=i\wedge u_e<0}|u_e| - \left(\sum_{e:j_e=i\wedge u_e<0}|u_e| + \sum_{e:i_e=i\wedge u_e>0}|u_e|\right) = 1$.

The definition (18) of $u_e, e \in \partial E$ is used to prove the inequality (13):

$$\sum_{e:i_e=0} u_e = \sum_{e\in\partial E} u_e = u_{(0,p)} + u_{(0,q)} = |V_1^*| + |V_2^*| = n$$

Since T_1 and T_2 are the spanning trees of G_1 and G_2, then $\sum_{e\in E} y_e = |E'_1| = |V_1| - 1$ and $\sum_{e\in E} z_e = |E'_2| = |V_2| - 1$. That implies $\sum_{e\in E} y_e + \sum_{e\in E} z_e = |V_1| + |V_2| - 2 = n - 2$. Thus, the constraint (14) is proved.

For $e \in \partial E,\ y_e = 1$ for only one e, and that is the case when $e = (0,p)$. For all the other edges $e \in \partial E$, $y_e = 0$, which implies $\sum_{e\in\partial E} y_e = 1$. Similarly, $\sum_{e\in\partial E} z_e = 1$, implying that the constraint (15) is satisfied.

($\Leftarrow$). Suppose that optimal solution (x^*, y^*, z^*, u^*) satisfies the conditions (4)–(17). The partition (V_1, V_2) will be constructed, so that $obj(V_1, V_2) \leqslant obj_{MILP}(x^*, y^*, z^*, u^*)$.

Let us define.

$V_1 = \{i \in V : x_i = 1\},\ \bar{E}'_1 = \{e \in E : y_e = 1\}$ and
$V_2 = \{i \in V : x_i = 0\},\ \bar{E}'_2 = \{e \in E : z_e = 1\}$

Constraints (6)–(9) ensures that the sets $\bar{E}'_1$ and $\bar{E}'_2$ are well defined, i.e. all edges from $\bar{E}'_1$ have endpoints from $V_1 \cup \{0\}$ and all edges from $\bar{E}'_2$ have endpoints from $V_2 \cup \{0\}$:

(i) $e \in E'_1$ implies $y_e = 1$. From the constraint (6) and the binary nature of the variables x_{i_e} and x_{j_e}, it follows that $x_{i_e} = 1$ and $x_{j_e} = 1$, which implies that $i_e,\ j_e \in V_1$.
(ii) $e \in \partial E \cap \bar{E}'_1$ also implies $y_e = 1$. From the constraint (8), $x_{j_e} = 1$ follows, which implies that $j_e \in V_1$.

Similarly, constraints (7) and (9) ensure that all edges from $\bar{E}'_2$ have endpoints from $V_2 \cup \{0\}$. Constraint (14) ensures that the total number of edges included in spanning trees is exactly $n - 2$.

It has already been shown that $w(V_1) - w(V_2) = -w_{sum} + 2\sum_{i=1}^{n} x_i w_i$. From the constraint (5) it follows that $w(V_1) - w(V_2) \geqslant 0$, so the inequality $|w(V_1) - w(V_2)| \leqslant -w_{sum} + 2\sum_{i=1}^{n} x_i w_i$ holds.

From inequalities (10) and (11), it follows that $u_e = 0$, for all $e \notin \bar{E}'_1 \cup \bar{E}'_2$.

Now it can be proven that the graphs (V_1, E'_1) and (V_2, E'_2) are connected. The proof for the graph (V_1, E'_1) is given and the connectivity of the second graph is proven similarly.

Suppose that S' and S'' are arbitrary subsets of V_1, such that, $S' \cup S'' = V_1$ and $S' \cap S'' = \emptyset, S', S'' \neq \emptyset$. Then $(\exists e \in E'_1),\ i_e \in S' \wedge j_e \in S''$ will be proved.

Let us summarize the constraints given in (12), for all $i \in S'$. We get

$$\sum_{i\in S'}\left(\sum_{e:j_e\in S'} u_e - \sum_{e:i_e\in S'} u_e\right) = |S'|.$$

If the sum from the left side is disassembled, the following expression is gotten:

$$\sum_{i\in S'}\left(\sum_{e:j_e\in S'} u_e - \sum_{e:i_e\in S'} u_e\right) = \sum_{e:j_e\in S'\wedge i_e\in S'} u_e + \sum_{e:j_e\in S'\wedge i_e\in S''} u_e + \sum_{e:j_e\in S'\wedge i_e=0} u_e + \sum_{e:j_e\in S'\wedge i_e\in V_2} u_e - \left(\sum_{e:i_e\in S'\wedge j_e\in S'} u_e + \sum_{e:i_e\in S'\wedge j_e\in S''} u_e + \sum_{e:i_e\in S'\wedge j_e\in V_2} u_e\right).$$

Let us denote summands in the last equation as $A,\ B,\ C,\ D,\ E,\ F$ and G, the equation can be written as

$$\sum_{i\in S'}\left(\sum_{e:j_e\in S'} u_e - \sum_{e:i_e\in S'} u_e\right) = A + B + C + D - (E + F + G)$$

It is obvious that $A = E$ and $D = G = 0$ (between V_1 and V_2 there are no edges in E'). Further, $C = 0$ or $C = |V_1|$ depending on whether $p \in S'$ or $p \notin S'$, because the Lemma 1 proposes that exactly one vertex (p) from V_1 is connected to the vertex 0. Thus, we have

$$\sum_{e:j_e\in S'\wedge i_e\in S''} u_e \neq \sum_{e:i_e\in S'\wedge j_e\in S''} u_e$$

The last inequality implies $(\exists e)\ (((i_e \in S') \wedge (j_e \in S'')) \vee ((i_e \in S'') \wedge (j_e \in S')))\ u_e \neq 0$. From $u_e \neq 0 \Rightarrow y_e = 1 \Rightarrow e \in E'_1$ and the edge connecting S' and S'' is found. Thus, the component (V_1, E'_1) is connected.

Therefore, constructed partition (V_1, V_2) is connected and $obj(V_1, V_2) \leqslant obj_{MILP}(x^*, y^*, y^*, u^*)$. □

Example 2. Let G be a graph given in the Example 1. Then the optimal solution, after solving MILP (4)–(17) by CPLEX is: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1,\ x_6 = 0$ Values of y, z, u are given in Table 1 and the solution is shown in Fig. 2.

As the Lemma 1 proposes, the extra vertex is connected to exactly one vertex per component. If the introduced notation is still held, for the first component, vertex p is vertex 5, and for the second one, vertex q is vertex 6. Theorem 1 proposes that the flows on the edges $(0, p)$ and $(0, q)$ are equal to $|V_1|$ and $|V_2|$: the flows for the edges $(0, 5)$ and $(0, 6)$ are both equal to 3, because each component has 3 vertices. Each edge in the spanning trees has a non-zero flow, and the sign of the flow depends on the orientation of the edge. Flows on edges not belonging to spanning trees are equal to zero. Although the graph is rather small, it can be seen that the flow (absolute value) on each edge (i, j) in the spanning trees is equal to the number of vertices in the subtree, rooted by the vertex j.

Example 3. Let G be a grid graph, given in Fig. 3 (left). Grid graph of dimension $n = rxs$ is a graph with the vertices corresponding to the points in the plane with integer coordinates. Two vertices are connected by an edge whenever the corresponding points are at the distance 1. The interior vertices of the grid graph have 4 neighbors, the vertices along the sides have 3 neighbors and only four of them have 2 neighbors (extreme vertices of the rectangle). Hence, the grid graphs are sparse, containing $(2\ rs - r - s)$ edges. A weighted grid graph of the dimension 5x5 is shown in Fig. 3 (left). It contains 25 vertices and 40 edges. One solution is shown in Fig. 3 (right). In order to unburden the Fig. 3 (right), vertex weights and edge orientation are omitted and the sign of the flow indicates the orientation. As it can be seen, the extra vertex 0 is connected to vertices 12 (vertex p) and 14 (vertex q). The flows on the edges $(0, p)$ and $(0, q)$ are equal to 12 and 13, which indicates that the components in the partition have 12 and 13 vertices. As it is stated in the proof of the Theorem 1, the flow (the absolute value) on each edge (i, j) in the spanning trees is equal to the number of vertices in the subtree, rooted by the vertex j. If the sums of vertex weights in each component are calculated, it can be seen that the difference between these two values is equal to $|630 - 629| = 1$. This fact proves that the solution shown in Fig. 3 (right) is optimal, because the total sum of all weights is odd and the objective function is non-negative.

Table 1
Values of y, z, u calculated by CPLEX.

Edge	1–2	1–6	2–3	2–4	2–6	3–4	4–5	4–6	5–6	0–1	0–2	0–3	0–4	0–5	0–6
y	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
z	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1
u	−1	0	0	0	−2	−1	−2	0	0	0	0	0	0	3	3

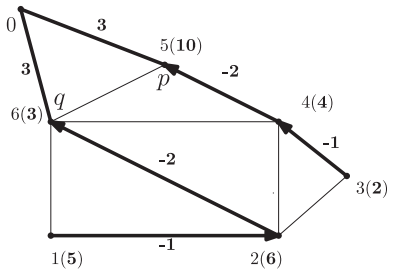

Fig. 2. The solution of the graph given in Example 1.

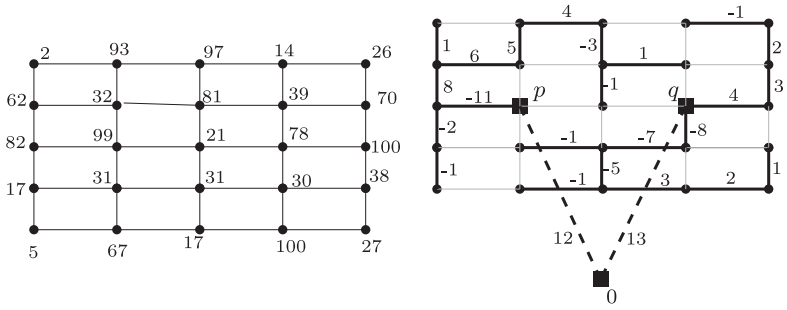

Fig. 3. The grid graph 05x05 (left); the partition obtained by the MILP solver (right).

Initialization:

1. Determine the initial solution x (usually at random).
2. Select the set of neighborhood structures N_k, for $k = k_{min}, ..., k_{max}$, that will be used in the search;
3. Select a stopping criterion.

While the stopping criterion is not met do

1. Set $k := k_{min}$
2. While $k <= k_{max}$ do

Shaking procedure. Generate a point x' at random from the k-th neighborhood of x ($x' \in N_k(x)$);

Local Search procedure. Apply a local search method with x' as initial solution;

Denote x'' the obtained local optimum;

Move or not procedure. If this local optimum is better than the incumbent, move

there ($x := x''$) and go to Step (1); Otherwise, set $k := k + 1$.

Collect output data and print the solution

Fig. 4. VNS algorithm scheme.

3. VNS for solving MBCP

The VNS is an effective metaheuristic approach introduced by Mladenović and Hansen in [13]. VNS proceeds to a systematic change of neighborhoods in order to avoid the algorithm traps in local suboptimal solutions. Efficiency of the VNS approach is based on the observation that in many practical optimization problems multiple local minima are in some kind of correlation. Thus, the idea of directing the search from one to another (perhaps better) local minimum through their neighborhoods is a reasonable approach. A detailed description of different VNS variants is out of the paper's scope and can be found in [14,15]. Recent successful VNS implementations are:

- Ready-mix concrete delivery problem [16];
- Harmonic means clustering [17];

- Finite-horizon Markov Decision Processes [18];
- 0–1 mixed integer programming [19].

In order to disseminate the search, the VNS uses the neighborhoods of increasing cardinality, seeking for better local optimum during the local search. To define a set of neighborhood structures, suppose that x is an arbitrary solution and N_k, for $k = k_{min}, \ldots, k_{max}$, a finite set of pre-selected neighborhood structures. Then $N_k(x)$ is defined as the set of solutions in kth neighborhood of x. The stopping criteria most often used are maximum number of iterations, maximum number of iterations between two improvements or maximum CPU time allowed.

The basic scheme of VNS is shown in Fig. 4.

The whole VNS is realized through two nested loops. The outer loop works as a diversifier reiterating the inner loop, while the inner loop carries the major search via two main functions, shaking and local search. Local search is trying to improve the solution within the local neighborhood, whilst shaking diversifies the solution by switching to another local neighborhood. The inner loop iterates as long as it keeps improving the solutions. Once an inner loop is completed, the outer loop re-iterates until the stopping criterion is met.

3.1. Initialization and objective function

In the VNS for solving MBCP, the solution for the given graph $G = (V, E)$ is represented as a binary array x with the length $|V|$. The elements of the array correspond to vertices, and indicate in which of two subsets of V vertices are arranged, i.e. $i \in V_1$ if $x_i = 1$ and $i \in V_2$ if $x_i = 0$. Initial solution is chosen randomly.

During the searching process, infeasible solutions can appear, which means that one (or both) component(s) induced by V_1 and V_2 (G_1 and G_2 respectively) is (are) disconnected. Such solutions are incorrect and the VNS has to be oriented towards the correct solution space. A standard method for overcoming the situations when infeasible solutions appear is the usage of the penalty function. Generally, the value of the penalty function should be a bit greater than the minimal correction value needed for correcting the infeasible solution to the feasible one. Contrary, if the value of the penalty function is less than the minimal correction value, the infeasible solutions can appear as the final result of the search, which is not allowed. At the other hand, if the value of the penalty function is significantly greater than the minimal correction value, infeasible solutions are totaly discriminated. That is not good, because sometimes high quality solutions are easier achieved through the "good" infeasible solutions rather than the "bad" feasible ones.

In the case of MBCP, it is convenient to incorporate the total number of connected components into the formulation of the penalty function. This number should be multiplied with an appropriate value, which depends on concrete graph. The experiments show that maximal vertex weight should be convenient to use. So, to formulate the penalty function for the MBCP, for each subgraph generated by the VNS, a check of the number of connected components is performed. For this task, the breadth-first search algorithm (BFS) is used. In the case when the BFS finds unconnected component (s), the penalty function is applied. So, if $nc(G_1)$ and $nc(G_2)$ are the numbers of components of G_1 and G_2 respectively, the penalty function f_{pen} is calculated by the formula

$$f_{pen} = (nc(G_1) + nc(G_2) - 2) * w_{max} \tag{19}$$

where w_{max} is the maximal weight in the graph.

For the current solution, the algorithm calculates the objective value by the formula

$$obj(V_1, V_2) = |w(V_1) - w(V_2)| + f_{pen} \tag{20}$$

```
function Obj(S)
    sum1 := 0, sum2 := 0;
    nc1 := 0, nc2 := 0;
    for i ∈ V do
        if x[i] = 1
            then
                sum1 := sum1 + w[i];
            else
                sum2 := sum2 + w[i];
        endif
    endfor
    nc1:=ncomp(1);
    nc2:=ncomp(2);
    return abs(sum1 − sum2) + (nc1 + nc2 − 2) ∗ w_max
```

Fig. 5. Pseudo-code for the objective function.

From (19) it is obvious that if G_1 and G_2 are connected, then the penalty function is equal to 0, because $nc(G_1) = nc(G_2) = 1$. In that case, the penalty function has no influence on the objective value (20). In cases when at least one component is disconnected, the penalty function significantly increases the objective value, which means that infeasible solutions can not be held as the best for a long time. The fact that the achieved final solutions are feasible indicates that the value of the penalty function is greater than the minimal correction value. Also, according to good experimental results, it seems that the penalty function is not too rigid.

Fig. 5 presents the pseudo-code of the objective function of the current solution S. For each $i, i \in V$, the values $x[i]$ denote the component of the vertex i. The value w_{max} denotes the maximum weight and is calculated earlier, during the initialization phase. The values $sum1$ and $sum2$ are calculated as the total sum of weights for vertices from V_1 and V_2 and the variables $nc1$ and $nc2$ are calculated in the function ncomp (), which implements standard BFS algorithm for finding the number of connected components.

Time complexity of the objective function is calculated as follows. Time complexity for summing of all weights in the corresponding subsets is $O(|V|)$. Time complexity for the function ncomp is equal to time complexity for the BFS algorithm which is $O(|V_1| + |E_1|)$ for the first and $O(|V_2| + |E_2|)$ for the second component. Thus, the overall time complexity of the calculation of the objective function is $O(|V| + |E|)$.

3.2. Neighborhoods and the shaking procedure

The shaking procedure creates a new solution x', $(x' \in N_k(x))$ based on the current best solution x. The kth neighborhood is defined as follows. Some k vertices from V are chosen randomly. For each chosen vertex, the component is changed: all chosen vertices belonging to V_1 are moved into V_2 and vice versa. Formally, the kth neighborhood of the vector x can be written as $N_k(x) = \{x' : \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, |V|\}\ x'_{i_j} = 1 - x_{i_j}\}$.

In the algorithm, the value k_{min} is set to 2, and k_{max} is defined as $k_{max} = \min\{30, |V|/2\}$. The reason for this is that k_{max} is trying to satisfy the theoretical condition which states that the size of each neighborhood should be greater than the size of the previous one. Since the size of the k- the neighborhood is $\binom{|V|}{k}$, for $k < |V|/2$ implies $\binom{|V|}{k-1} < \binom{|V|}{k}$, i.e. $|N_{k-1}(x)| < |N_k(x)|$ and the condition is satisfied. For larger instances, experiments show that $k_{max} = 30$ is enough for reaching good quality solutions.

It is clear that shaking consists of k steps with $O(|V|)$ time complexity, so the overall time complexity of the calculation of the Shaking () procedure is $O(k_{max} \cdot |V|)$.

3.3. Local search

For the solution x' obtained by the shaking procedure, the local search is called. In each iteration of the local search, the algorithm swaps the components of two vertices, which belong to different components. For example, if $u \in V_1$ and $v \in V_2$, after the swapping the status is $u \in V_2$ and $v \in V_1$. The swapping forms a new solution, denoted as x''. In the case when x'' is better solution than x', x' becomes equal to x'', (otherwise, x' is not changed) and the local search continues with the next pair of vertices. The local search stops, when the solution can no longer be improved. If the objective value of x' is strictly greater than the objective value of the incumbent, the search is repeated with the same x and the next neighborhood. In the case when the objective value of x' is less than of the x, the currently best solution x gets the value x'. If the objective values of the two solutions x and x' are the same, then $x = x'$ is set with probability p_{move} and the algorithm continues the search with the same neighborhood. In other case, a search is repeated with the same x and the next neighborhood with probability $1 - p_{move}$.

It is obvious that the local search forms pairs of vertices and the total number of them is $O(|V|^2)$. Swapping the component for each pair is done in $O(1)$, and the calculation of the objective value of the new solution is done in $O(|V| + |E|)$. So, the overall time complexity of the calculation of the local search is $O(|V|^3 + |E| \cdot |V|^2)$.

After all neighborhoods have been considered, the algorithm again begins with the first, until the stopping criterion is satisfied. In our case, the stopping criterion is the maximum number of iterations reached.

4. Computational results

This section presents the computational results, which show effectiveness of the proposed MILP formulation and the VNS method. All tests are carried out on the Intel Core 2 Quad Q9400 @2.66 GHz with 8 GB RAM. For testing the MILP formulation, two MILP solvers are used: CPLEX 12.1 solver and Gurobi 4.0 solver. For both solvers, computational time is limited to 7200 s. The VNS implementation is coded in C programming language. Since the VNS has a nondeterministic nature, it is executed 20 times with different random seeds. For each instance, k_{min} is set to 2. In order to ensure that $(\forall k)(k_{min} \leqslant k \leqslant k_{max} - 1)$ $|N_k(x)| \leqslant |N_{k+1}(x)|$, for instances with less than 60 vertices k_{max} is set to $n/2$ and for larger instances k_{max} is set to 30. The parameter p_{move} is set to 0.4. The algorithm stops after 500 iterations. To make a fair testing of both MILP solvers and the VNS method, only one core of the processor per each MILP execution is used.

For all experiments two sets of instances are used: grid graph instances and random generated instances. Grid graph instances were introduced in [20], and were used as test instances for the k-cardinality tree problem (KCTP). Random graph

instances were generated in [2]. Both the grid graph and random instances were used for testing GA for solving the MBCP in [2]. The set of grid graph consists of 16 instances: the smallest has 25 vertices, ($n = 05x05$), and the greatest has 225 ($n = 15x15$) vertices. For the instances with names which end with letter (a), the vertex weights are integers, chosen from a uniform distribution in the interval [1100] and in the second case (instance name ends with letter b), random interval chosen is [1500]. The first instance, named gg_05_05a is exactly the instance shown and analyzed in Example 3.

The second set of instances consists of 21 connected random graphs with real vertex weights, chosen from a uniform distribution in the interval (0, 100].

Tables 2,3 provide computational results for the grid graph and random instances. In the first three columns, the instance names, the number of vertices and edges are given, respectively. The next column contains the optimal solution, with the mark '-' if the optimal solution is not known. The next two columns are related to MILP solvers CPLEX and Gurobi. The data in these columns are twofold: in the case when MILP solvers succeed in finding the solution in less than two hours (7200 s), the execution time is shown. In the other case, when MILP solver can not find the optimal solution in 7200 s, two subcases are considered: a solution is found but not verified as optimal and another, MILP solver can not achieve the solution at all. In the first of the two subcases, the value of the solution with the mark (n.v.) is shown, indicating that the solution is not verified as optimal in 7200 s and in the second subcase, when MILP solver could not find any solution, the mark N/A is used. The next two columns contain the results and the total execution time of the GA given in [2], with the mark *opt* in the result column (*GA*), if the optimal solution is reached. The last two columns contain data of the proposed VNS: the best achieved value, with the mark *opt* if the optimal value is reached and the average total time of VNS execution.

It can easily be shown that the optimal solution for the grid graph instances can not be less than 0, if the sum of all weights is even, and 1 if the sum of all weights is odd, because the value of the objective function is non-negative and all weights are integers. So, for these instances, the values achieved by the metaheuristics are optimal.

From Table 2 it is evident that the MILP solvers can be used to find the optimal solutions for smaller instances. In the case of the grid graph instances, both solvers achieve optimal solutions for most instances up to 70 vertices: CPLEX achieves 8 (of 10) while Gurobi achieves 7 (of 10) optimal solutions. For large grid graph instances, both solvers could not find any solution at all, which is indicated with the mark N/A.

From Table 3 it can be seen that Gurobi is more successful in finding optimal solutions for random instances than CPLEX. For instances up to 50 vertices, Gurobi finds all optimal solutions except one, while CPLEX finds the optimum only for the instances with 20 vertices and for one instance containing 30 vertices. Also, the computational time of Gurobi executions is less than the computational time of CPLEX. As expected, MILP solvers could not achieve optimal solutions in reasonable time for large random instances, but for all these instances CPLEX succeeds in finding the solutions which are not verified as optimal in 7200 s. Gurobi is less successful and finds only one such solution.

The VNS algorithm easily reaches all optimal solutions for the grid graph instances, as can be seen from Table 2. Also, the optimal solution is reached in each of the 20 runs, which indicates high reliability of the VNS algorithm. The computational time is rather small, and goes up to 216 s for the largest instance.

For the first two random graph instances (rnd01 and rnd02 from Table 3), both GA and VNS reach optimal solutions. For all other instances, the proposed VNS achieves very good results and outperforms the GA in all these cases. As it can be seen, the objective values achieved by the VNS are less than 10^{-3} (except three) and for very large instances, with 100 and more vertices and 300 and more edges, objective values of the solutions achieved by the VNS are less than 10^{-4}. For the two instances (rnd17 and rnd21), the achieved result is equal to 0, which is surely optimal, because the objective value is non-negative. So, results presented in Tables 2, 3 clearly indicate that the VNS approach can be successfully applied on large-scale problems, in cases where the exact methods fail.

Table 2
Computational results for the grid graph instances.

Inst.	$\|V\|$	$\|E\|$	Opt	$t_{cplex}(s)$	$t_{gur}(s)$	GA	t_{GA}	VNS	$t_{VNS}(s)$
05x05a	25	40	1	11.26	1016.99	opt	0.36	opt	0.34
05x05b	25	40	1	56.41	536.27	opt	0.37	opt	0.33
05x06a	30	49	0	4.11	0.17	opt	0.41	opt	0.50
05x06b	30	49	1	101.58	7077.59	opt	0.43	opt	0.60
05x10a	50	85	1	866.49	2539.09	opt	0.72	opt	2.30
05x10b	50	85	0	N/A	N/A	opt	0.91	opt	2.47
05x20a	100	175	0	N/A	N/A	opt	1.73	opt	18.12
05x20b	100	175	1	N/A	N/A	opt	1.77	opt	20.81
07x07a	49	84	0	3139.73	2528.21	opt	0.804	opt	2.79
07x07b	49	84	1	1053.79	N/A	opt	0.743	opt	3.31
07x10a	70	123	1	N/A	1858.10	opt	1.249	opt	6.38
07x10b	70	123	0	6112.54	N/A	opt	1.186	opt	7.33
10x10a	100	180	1	N/A	N/A	opt	1.809	opt	22.40
10x10b	100	180	1	N/A	N/A	opt	1.633	opt	25.05
15x15a	225	420	0	N/A	N/A	opt	4.439	opt	195.49
15x15b	225	420	0	N/A	N/A	opt	4.669	opt	216.16

Table 3
Computational results for the random graph instances.

Inst.	$\|V\|$	$\|E\|$	Opt	t_{cplex}	t_{gur}	GA	t_{GA}	VNS	t_{VNS}
rnd01	20	30	1.14274	500.13	187.51	opt	0.52	opt	0.159
rnd02	20	50	0.01965	1899.18	480.89	opt	0.49	opt	0.198
rnd03	20	100	0	1307.85	36.10	0.00626	0.68	0.0008	0.281
rnd04	30	50	0	n.v:0.1817	5510.0	0.00915	0.58	0.00241	0.522
rnd05	30	70	0	5361.1	2081.6	0.01036	0.65	0.00148	0.505
rnd06	30	200	0	n.v:0.0108	403.56	0.00029	0.93	0.00007	1.036
rnd07	50	70	–	n.v:0.0328	n.v:0.4176	0.01608	0.92	0.00104	3.282
rnd08	50	100	0	n.v:0	5188.6	0.00531	1.10	0.00005	2.386
rnd09	50	400	0	n.v:0.0484	5112.0	0.00024	1.92	0.00008	5.864
rnd10	70	100	–	n.v:0.2083	N/A	0.01328	1.40	0.00022	7.678
rnd11	70	200	–	n.v:0	N/A	0.00023	1.737	0.00001	8.140
rnd12	70	600	–	n.v:0.0356	N/A	0.00232	2.910	0.00004	13.202
rnd13	100	150	–	n.v:0	N/A	0.00859	2.032	0.00037	23.473
rnd14	100	300	–	n.v:0	N/A	0.00137	2.355	0.00001	22.984
rnd15	100	800	–	n.v:0.7812	N/A	0.00155	3.812	0.00001	46.39
rnd16	200	300	–	n.v:0	N/A	0.00574	4.972	0.00006	203.31
rnd17	200	600	–	n.v:0	N/A	0.00058	5.255	0	206.34
rnd18	200	1500	–	n.v:0	N/A	0.00011	7.380	0.00001	411.47
rnd19	300	500	–	n.v:0	N/A	0.00039	7.747	0.00001	692.21
rnd20	300	1000	–	n.v:0	N/A	0.00025	7.560	0.00001	609.02
rnd21	300	2000	–	n.v:0	N/A	0.00008	11.61	0	1256.5

Comparing the execution times of the two heuristics, it is obvious that the GA founds the solutions faster, but the VNS provides higher solution quality.

5. Conclusion

This paper is devoted to the Maximally Balanced Connected Partition Problem. A mixed integer linear programming formulation is introduced and the correctness of the corresponding formulation is proved. The presented model uses polynomial number of variables and constraints, which indicates that this model can be used both in theoretical and practical considerations.

The paper also introduces the VNS metaheuristic for solving MBCP. The suitable neighborhoods are implemented, based on changing the component for an increasing number of vertices. The VNS implements an efficient and relatively fast local search, which improves the current solution by swapping the component for pairs of vertices.

In order to prove the usability and reliability of the proposed methods, numerical experiments are carried out. The MILP model is run in two MILP solvers, CPLEX 12.1. and Gurobi 4.0. Experimental results show that the CPLEX and/or Gurobi implementations of the MILP model can find optimal solutions for most of the smaller and medium instances.

According to the computational results, the applied VNS approach proves to be successful. The VNS reaches most optimal solutions for smaller instances and provides good results for large-scale instances. This indicates that the VNS approach can be used reliably for solving the MBCP.

This research can be extended in several ways. The proposed MILP formulation and the VNS technique could be extended for solving the BCPq problem, for arbitrary q. Also, it would be useful to build the exact methods based on the proposed MILP formulation. Other heuristic methods and their hybridization with existing VNS implementation could also be investigated.

Acknowledgment

This research is supported in part by the Ministry of Science and Technology of Republika Srpska, Bosnia and Herzegovina (Research Project: ‘Approximations of matrix functions and their applications in Network Science’).

References

[1] J. Chlebikova, Approximating the maximally balanced connected partition problem in graphs, Inf. Process. Lett. 60 (1996) 225–230.
[2] B. Djurić, J. Kratica, D. Tosić, V. Filipović, Solving the maximally balanced connected partition problem in graphs by using genetic algorithm, Comput. Inf. 27 (3) (2008) 341–354.
[3] F. Chataigner, L.R.B. Salgado, Y. Wakabayashi, Approximation and inaproximability results on balanced connected partitions of graphs, Discrete Math. Theor. Comput. Sci. 9 (2007) 177–192.
[4] T. Ito, X. Zhou, T. Nishizeki, Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size, J. Discrete Algorithms 4 (2006) 142–154.
[5] T. Ito, T. Uno, X. Zhou, T. Nishizeki, Partitioning a weighted tree to subtrees of almost uniform size, LNCS, vol. 5369/2008, 2008, pp. 196–207.

[6] I. Slama, B. Jouaber, D. Zeghlache, Topology control and routing in large scale wireless sensor networks, Wireless Sens. Netw. 2 (2010) 584–598.
[7] M. Lucertini, Y. Perl, B. Simeone, Most uniform path partitioning and its use in image processing, Discrete Appl. Math. 42 (2–3) (1993) 227–256.
[8] B. Bozkaya, E. Erkut, G. Laporte, A tabu search heuristic and adaptive memory procedure for political districting, Eur. J. Oper. Res. 144 (1) (2003) 12–26.
[9] T. Assunção, V. Furtado, A heuristic method for balanced graph partitioning: an application for the demarcation of preventive police patrol areas, in: Advances in Artifical Intelligence – Iberamia 2008, in: H. Geffner, R. Prada, I.M. Alexandre, N. David (Eds.), LNCS, vol. 5290/2008, Springer, Berlin/Heidelberg, 2008, pp. 62–72.
[10] J. Kojić, Integer linear programming model for multidimensional two-way number partitioning problem, Comput. Math. Appl. 60 (8) (2010) 2302–2308.
[11] K. Afshar, M. Ehsan, M. Fotuhi-Firuzabad, N. Amjady, Cost-benefit analysis and MILP for optimal reserve capacity determination in power system, Appl. Math. Comput. 196 (2) (2008) 752–761.
[12] A. Savić, J. Kratica, M. Milanović, Dj. Dugošija, A mixed integer linear programming formulation of the maximum betweenness problem, Eur. J. Oper. Res. 206 (3) (2010) 522–527.
[13] N. Mladenović, P. Hansen, Variable neighbourhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[14] P. Hansen, N. Mladenović, J.A. Moreno-Pérez, Variable neighbourhood search: methods and applications (invited survey), 4OR, Q. J. Oper. Res. 6 (2008) 319–360.
[15] P. Hansen, N. Mladenović, J.A. Moreno-Pérez, Variable neighbourhood search: algorithms and applications, Ann. Oper. Res. 175 (2010) 367–407.
[16] V. Schmid, K.F. Doerner, R.F. Hartl, J.J.S. González, Hybridization of local branching guided by variable neighborhood search for ready-mixed concrete delivery problems, Comput. Oper. Res. 37 (3) (2010) 559–574.
[17] A. Alguwaizani, P. Hansen, N. Mladenović, E. Ngai, Variable neighbourhood search for harmonic means clustering, Appl. Math. Model. 35 (2011) 2688–2694.
[18] Q.H. Zhao, J. Brimberg, N. Mladenović, A variable neighborhood search based algorithm for finite-horizon Markov decision processes, Appl. Math. Comput. 217 (7) (2010) 3480–3492.
[19] J. Lazić, S. Hanafi, N. Mladenović, D. Urošević, Variable neighbourhood decomposition search for 0–1 mixed integer programs, Comput. Oper. Res. 37 (6) (2010) 1055–1067.
[20] C. Blum, M. Ehrgott, Local search algorithms for the k-cardinality tree problem, Discrete Appl. Math. 128 (2–3) (2003) 511–540.