Научном већу

Института за физику у Београду

Предмет: Молба за избор у звање истраживач сарадник

МОЛБА

Молим научно веће Института за физику у Београду да покрене поступак за мој избор у звање **Истраживач сарадник**, имајући у виду да испуњавам све критеријуме прописане од стране Министарства просвете, науке и технолошког развоја Републике Србије за стицање наведеног звања.

У прилогу достављам:

1. Мишљење руководиоца са предлогом комисије за избор у звање;
2. Биографију и преглед научне активности;
3. Списак и копије објављених научних радова;
4. Фотокопију диплома са основних и мастер студија;
5. Потврду о уписаним докторским студијама;
6. Потврду о прихватању теме докторске дисертације.

С поштовањем

У Београду, 04.05.2022.

[illegible]

....

Марија Ивановић

Научном већу

Института за физику у Београду

Предмет: Мишљење руководиоца за избор Марије Ивановић у звање истраживач сарадник

Марија Ивановић је завршила основне и мастер студије на Математичком факултету Универзитета у Београду – смер *Нумеричка математика и оптимизација*. На докторским студијама на Математичком факултету Универзитета у Београду, положила је све испите предвиђене планом и програмом и предала докторску тезу. Тема њене докторске дисертације представњена је и прихваћена од стране Научно-наставног већа Математичкког факултета Универзитета у Београду.
Од децембра 2021. године запослена је у Институту за физику у Београду, где је изабрана у звање истраживач приправник. Тренутно је ангажована на развијању и усавршавању програмских пакета који се користе у моделовању процеса који се одвијају у ткиву током микроталасне аблације, што је веома значајно за медицинске примене. Марија Ивановић је била ко-аутор на више радова публикованих у међународним часописима и учествовала је на више међународних конференција.
С обзиром да Марија Ивановић испуњава све услове предвиђене Правилником о поступку и начину вредновања и квантитативном исказивању научноистраживачких резултата истраживача, сагласна сам са покретањем поступка за њен избор у звање истраживач сарадник.

За чланове комисије, за избор Марије Ивановић у звање истраживач сарадник, предлажем следећи састав:
1. др Марија Радмиловић-Рађеновић, научни саветник, Институт за физику Београд
2. др Бранислав Рађеновић, научни саветник у пензији, Институт за физику Београд
3. проф. др. Милан Дражић, редовни професор, Математички факултет Универзитета у Београду
4. проф. др. Александар Савић, ванредни професор, Математички факултет Универзитета у Београду

С поштовањем

Marija Radmilović-Radjenović

....

У Београду, 04.05.2022.

др Марија Радмиловић-Рађеновић
научни саветник

Биографија

Марија Ивановић је завршила основну школу „Јован Цвијић“ у Костолцу, док је Гимназију „Јован Шербановић“ завршила у Пожаревцу. На смеру *Нумеричка математика и оптимизација*, на Математичком факултету, дипломирала је 2007. године. На истом смеру је, септембра 2011. Године, одбранила мастер тему под називом „Теорија игара – игре сусретања и игре налажења“ и тиме стекла звање Мастер-математичар. Тренутно је студент докторских студија на Математичком факултету Универзитета у Београду (положених 154 ЕСПБа).

На Математичком факултету је од 2008.-2021. године држала наставу на више различитих предмета који се тичу нумеричке математике и оптимизације на основним и мастер студијама (Увод у нумеричку математику, Нумеричка математика IIа и IIб, Основи математичког моделирања, Методе коначних елемената, Диференцијалне једначине, Нелинеарно програмирање, Операциона истраживања, Теорија игара). Од 2009.-2021. године била је учесник на пројекту Министарства просвете и технолошког развоја под бројем ТР36006. Учествовала је на Темпус пројекту SEE Doctoral Studies in Mathematical Sciences 44703 - TEMPUS-1 -2008- 1 и на European Study Group with Industry ESGI 99 у организацији Природно-Математичког факултета Универзитета у Новом Саду. Од децембра 2021. године запослена је у Институту за физику у Београду, где је изабрана у звање истраживач приправник.

Научна активност

Током рада на Математичком факултету Универзитета у Београду, научна активност Марије Ивановић била је усмерена на развој математичких модела и хеуристика за њихово решавање. У оквиру докторских студија, развила је први математички модел за једну модификацију проблема римске доминације, док је сам проблем римске доминације оптимизовала развојем два нова математичка модела. Поред проблема римске доминације, уочила је неправилности које су постојале код модела који се односио на слабу римску доминацију, па је исту кориговала и унапредила увођењем новог модела.

Имајући у виду да се поменути проблеми примењују на графицима великих димензија, чинећи њихово егзактно решавање готово немогуће, Марија Ивановић је развила један модел којим се проблем римске доминације решава егзактно за једну специјалну врсту графова. Осим тога, развила је и две хеуристичке методе којима се проблем римске доминације и проблем слабе римске доминације могу решити приближно за било које графове. Експериментални резултати показали су предност развијених хеуристичких метода у односу на комерцијалне софтвере на графовима већих димензија. Поред поменутих проблема, један део истраживања односио се на развој математичког модела и хеуристичке методе за решавање проблема који се тичу динамичке алокације меморије.

Током рада у Институту за физику фокус истраживања преусмерен је на методу коначних елемената и њену примену за развој нумеричких модела који би се примењивали у биомедицини. Марија Ивановић је тренутно ангажована на развијању методологије за решавање једначине преноса топлоте у временском домену, као и Хелмхолцове једначине за електромагнетно поље методом коначхих елемената, које се користе за моделовање процеса који се одвијају у ткиву током микроталасне аблације, третмана који је веома значајан за отклањање тумора. Наиме, показало се да за сваки тумор, понаособ, постоје оптимални параметри (улазна снага, време аблације, итд.) који би омогућили што ефикаснији аблативни третман, уз отклањање целокупног туморалног ткива и минимално оштећење здравог ткива. Имајући у виду да се развој модела ткива применом методе коначних елемената одвија у више корака, укључујући дефинисање геометрије, постављање граничних услова, генерисање мреже, итд., познавање нумеричких метода кандидаткиње је од највеће важности.

Spisak referenci Marije Ivanović

Objavljeni radovi:

1. M. Ivanović, *„Improved integer linear programming formulation for weak roman domination problem”*, **Soft Computing**, ONLINE ISSN 1433-7479, strane 1-11, 2017. (**M_{22}**)
2. M. Ivanović, D. Urošević, *„Variable Neighborhood Search Approach for Solving Roman and Weak Roman Domination Problems on Graphs”*, **Computing and Informatics 38 (1)**, strane 57-84, 2019. (**M_{23}**)
3. M. Ivanović, *„A mixed integer linear programming formulation for restrained roman domination problem”*, **Theory of Applied Mathematics & Computer Science**, TAMCS, 5(2), strane 110-115, 2015. (**M_{24}**)
4. M. Ivanović, *„Improved mixed integer linear programing formulation for roman domination problem”*, **Publications de l'Institut Mathematique**, 99(113), strane 51-58, 2016. (**M_{24}**)
5. M. Ivanović, A. Savić, D. Urošević, Đ. Dugošija, *„A new Variable Neighborhood Search approach for solving Dynamic Memory Allocation Problem“*, **Yugoslav Journal of Operations Research**, Vol 28, No 3 (2018), p 291-314. (**M_{51}**)
6. Lian Chen, Huiqin Jiang, Yehui Shao, Marija Ivanović, *„Dominator and total dominator coloring in vague graphs“*, **Engineering and Applied Science Letters**, 2(2), 10-17, 2019, doi:10.30538/psrp-easl2019.0017 (**M_{24}**)

Saopštenje sa međunarodnog skupa štampano u celini M_{33}

1. M. Ivanović, Đ. Dugošija, A. Savić, D. Urošević, *„A New Integer Linear Formulation for a Memory Allocation Problem”*, Proceeding of XI Balkan Conference on Operational Research, pp. 284 – 288. Belgrade, Serbia, 2013.
2. M. Ivanović, D. Urošević, *„A New Linear-Time Algorithm for Computing the Weak Roman Domination Number of a Block Graph”*, Proceedings of XIII Balkan Conference on Operational Research, pp. 25-31. Belgrade, Serbia, May 25-28, 2018.

Saopštenje sa međunarodnog skupa štampano u izvodu M_{34}

1.M.Ivanović, D.Urošević, *Variable Neighborhood Search Solution for some variants of the Roman Domination problem*, 4th International VNS conference, Book of abstracts p.23 Malaga, Spain, October 305, 2016.
2. J.Brimberg, M. Ivanović, N.Mladenović, D.Urošević, *Primal-Dual VNS for large p-center problem*, ICVNS2018 – 6th International Conference on Variable Neighborhood Search, Sithonia, p.47. Halkidiki, Greece, October ,2018.
3. M. Radmilović-Rađenović, M. Ivanović, Branislav Rađenović, *The Effect of the Antenna Design on the Characteristics of Microwave Ablation Treatment on Liver Cancer*, MAS 16th International European Conference On Mathematics, Engineering, Natural & Medical Sciences, Mardin, Turkey, February 22-23, 2022, Conference program, pp.6.
4. M. Radmilović-Rađenović, N. Bošković, M. Ivanović, B. Rađenović, *Finite Element Analysis of the Efficiency of Multislot Antenna in Microwave Tissue Ablation*, 6-th Ankara International Congress on Scientific , Ankara, Turkey, April 1-3, 2022, Congress program, pp. 51.
5. M. Radmilović-Rađenović, M. Ivanović, N. Bošković, B. Rađenović, *Simulation studies of the effect of the input power on the performance of microwave tissue ablation*, International Scientific Research Congress, Baku, Azerbaijan, April 27-28, 2022, Congress program, pp. 31.

Zbirka zadataka

1. A. Delić, Z. Dražić, S. Živanović, M. Ivanović, *„Zbirka rešenih zadataka iz Uvoda u numeričku matematiku“*, izdavač: **Matematički fakultet**, Univerzitet u Beogradu, Srbija.

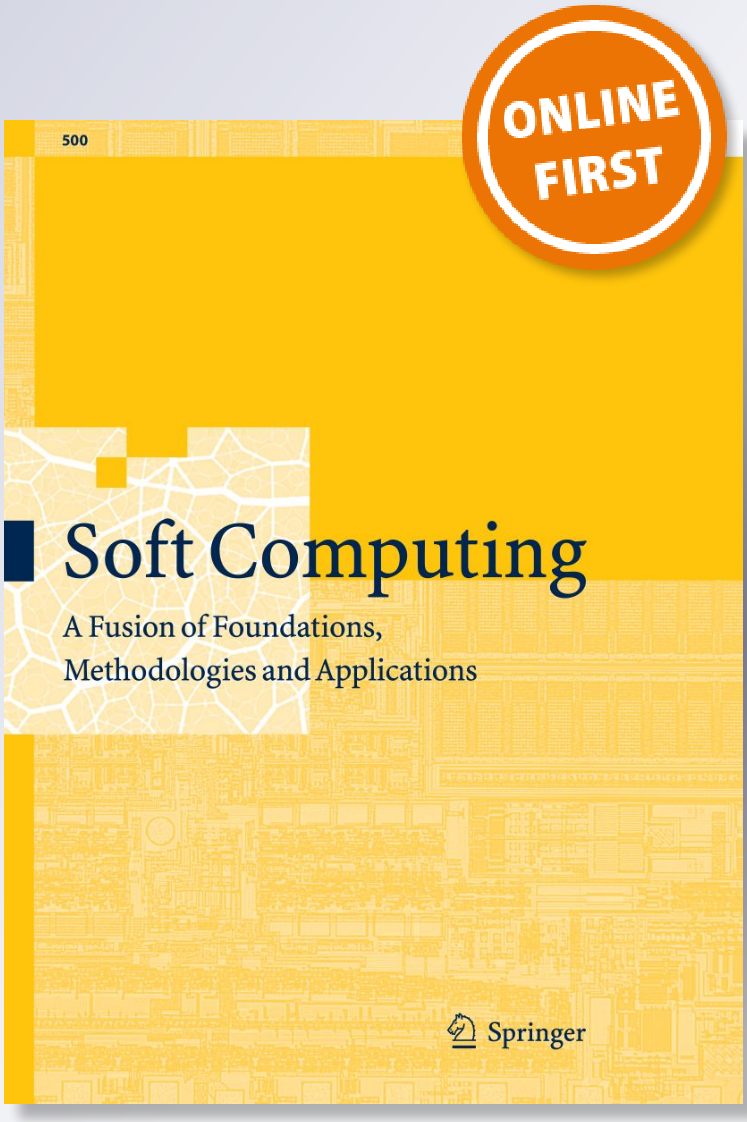
Improved integer linear programming formulation for weak Roman domination problem

Marija Ivanović

Soft Computing
A Fusion of Foundations,
Methodologies and Applications

ISSN 1432-7643

Soft Comput
DOI 10.1007/s00500-017-2706-4



Soft Comput
DOI 10.1007/s00500-017-2706-4

METHODOLOGIES AND APPLICATION

Improved integer linear programming formulation for weak Roman domination problem

Marija Ivanović[1]

Abstract Let $f : V \rightarrow \{0, 1, 2\}$ be a function, $G = (V, E)$ be a graph with a vertex set V and a set of edges E and let the weight of the vertex $u \in V$ be defined by $f(u)$. A vertex u with property $f(u) = 0$ is considered to be defended with respect to the function f if it is adjacent to a vertex with positive weight. Further, the function f is called a weak Roman dominating function (WRDF) if for every vertex u with property $f(u) = 0$ there exists at least one adjacent vertex v with positive weight such that the function $f' : V \rightarrow \{0, 1, 2\}$ defined by $f'(u) = 1$, $f'(v) = f(v) - 1$ and $f'(w) = f(w)$, $w \in V \setminus \{u, v\}$ has no undefended vertices. In this paper, an optimization problem of finding the WRDF f such that $\sum_{u \in V} f(u)$ is minimal, known as the weak Roman domination problem (WRDP), is considered. Therefore, a new integer linear programing (ILP) formulation is proposed and compared with the one known from the literature. Comparison between the new and the existing formulation is made through computational experiments on a grid, planar, net and randomly generated graphs known from the literature and up to 600 vertices. Tests were run using standard CPLEX and Gurobi optimization solvers. The obtained results demonstrate that the proposed new ILP formulation clearly outperforms the existing formulation in the sense of solutions' quality and running times.

Keywords Weak Roman domination in graphs · Combinatorial optimization · Integer linear programming

Communicated by V. Loia.

This research was partially supported by the Serbian Ministry of Education, Science and Technological Developments under the Grant No. TR36006.

Electronic supplementary material The online version of this article (doi:10.1007/s00500-017-2706-4) contains supplementary material, which is available to authorized users.

✉ Marija Ivanović
maria.ivanovic@gmail.com; marijai@math.rs

1 Faculty of Mathematics, Univeristy of Belgrade, Studentski Trg 16, Belgrade, Serbia

1 Introduction

Problem of domination is a very popular area in a graph theory. Inspired by the real-world historical problem, in his article "Defend the Roman Empire!" Stewart (1999), Ian Stewart introduced a Roman domination problem (RDP) as a problem of organizing Roman legions such that any province without legion stationed within it must be adjacent to a province with at least two legions stationed within it. Hence, in case of an attack, when two legions are stationed within one province, one legion is considered to be permanently stationed in order to keep that province safe, while the second legion could move in order to defend an adjacent province.

In order to keep the Roman Empire safe against one attack, several generalizations and variations followed. This paper is devoted to a one of them, known as weak Roman domination problem.

Weak Roman domination problem (WRDP) was proposed by Henning and Hedetniemi (2003) as an alternative approach for defending the Empire against a single attack and can be interpreted as follows. Given that every province without legion stationed within it is vulnerable to an attack, the WRDP requires that such a province must be adjacent to a province with at least one legion stationed within it, i.e., province is considered to be defended if there is a legion stationed within it or stationed within the adjacent province.

Published online: 10 July 2017

Furthermore, movement of a legion to an adjacent province which is without any legion stationed within it must not create an undefended province.

In a graph terminology, for a graph $G = (V, E)$ and a function f, $f : V \to \{0, 1, 2\}$, a vertex $u \in V$ with property $f(u) = 0$ is considered to be defended, with respect to f, if it is adjacent to at least one vertex $v \in N_u = \{v|(u, v) \in E\}$ with positive weight ($f(v) > 0$). Further, the function f is called a weak Roman dominating function (WRDF) for a graph G if for every vertex $u \in V$ such that $f(u) = 0$ there exists an adjacent vertex $v \in V$ such that $f(v) > 0$ and a function $f' : V \to \{0, 1, 2\}$, defined by $f'(u) = 1$, $f'(v) = f(v) - 1$ and $f'(w) = f(w)$, $w \in V \setminus \{u, v\}$ has no undefended vertices. The weight of the function f is calculated by a formula $w(f) = \sum_{u \in V} f(u)$ and the minimal weight among all WRDF f for a graph G is denoted by $\gamma_r(G)$. The WRDP can be described as a problem of finding the WRDF f for a graph G with the minimal weight.

However, the WRDP is not applicable to a war strategic problems only, and it can be also used for a huge number of recourse sharing problems. For instance, emergency vehicles are very expensive and minimizing their numbers, while still being able to help en injured people, will be of great use. Therefore, if emergency vehicles are organized between hospitals as Roman legions between their provinces, all hospitals will either own a vehicle or there will be en emergency vehicle in a neighborhood hospital that can be borrowed.

This paper is organized as follows. A previous work is given in Sect. 2, while the new improved integer linear programing formulation is presented in Sect. 3. Computational results are summarized in Sect. 4.

2 Previous work

Let $G = (V, E)$ be a simple undirected graph with a vertex set V and a set of edges E. A subset $V' \subseteq V$ is called dominating in a graph G, if every vertex in $V \setminus V'$ is adjacent to some vertex in V'. The cardinality of the minimum dominating set in G, denoted by $\gamma(G)$, is called the domination number of a graph G, and a problem of finding the set V' is known as the domination problem.

The function f, $f : V \to \{0, 1, 2\}$ defined such that every vertex $u \in V$ with property $f(u) = 0$ is adjacent to a vertex $v \in V$ with property $f(v) = 2$ is called a Roman dominating function (RDF). The weight of the function f is calculated by a formula $w(f) = \sum_{v \in V} f(v)$, and a problem of finding the RDF f with the minimal weight is called a Roman domination problem (RDP). The minimal weight among all RDF represents a Roman domination number, denoted by $\gamma_R(G)$.

The RDP was initially proposed by Stewart (1999) and ReVelle and Rosing (2000). Later, with the potential of saving the Empire substantial costs of maintaining legions while still defending the Roman Empire, several generalizations and improvements arose. Some of them are the Roman k-domination Henning (2003), the weak Roman domination Henning and Hedetniemi (2003), the restrained Roman domination Pushpam and Mai (2011), the edge Roman domination Chang et al. (2014), the secure Roman domination Burger et al. (2013), the total Roman domination Liedloff et al. (2005), the independent Roman domination Targhi et al. (2012), the signed Roman domination Ahangar et al. (2014) and the strong Roman domination Alvarez-Ruiz et al. (2015).

In this paper, the weak Roman domination problem (WRDP) is considered, and therefore, the previous work will be related only to that topic.

First, Henning and Hedetniemi (2003) observed that every RDF in a graph G is also a WRDF in a G and proved that for every graph G, $\gamma(G) \leq \gamma_r(G) \leq 2\gamma(G)$. Then, they proved that $\gamma_r(P_n) = \lceil 3n/7 \rceil$ for a path P_n, $n \geq 1$, noting that cost savings of the weak Roman domination over the Roman domination number for a mentioned path, $\gamma_R(P_n) - \gamma_r(P_n) = \lceil 2n/7 \rceil - \lceil 3n/7 \rceil$, is either $\lfloor 5n/21 \rfloor$ or $\lceil 5n/21 \rceil$ and showed that $\gamma_r(C_n) = \gamma_r(P_n)$ for cycles C_n when $n \geq 4$. And finally they characterized graphs G for which $\gamma_r(G) = \gamma(G)$ and forests G for which $\gamma_r(G) = 2\gamma(G)$.

Considering properties of domination, Roman domination, weak Roman domination and secure domination problems, Cockayne et al. (2005) studied four parameters which give the minimum number of legions required to protect the graph under the mentioned strategies and obtained the exact values for specific classes of graphs. Also, they gave characterization of secure dominating sets which are minimal. Setting $\gamma_s(G)$ as the secure domination number on a graph G, Cockayne et. al. first proved that inequalities

$$\gamma(G) \leq \gamma_r(G) \leq \begin{cases} \gamma_R(G) \leq 2\gamma(G) \\ \gamma_s(G) \end{cases}$$

hold. Then, for a complete graph K_n, they gave proposition that $\gamma(G) = \gamma_r(G) = \gamma_s(G) = 1$, while $\gamma_R(G) = 2$ and for a complete bipartite graph $K_{p,q}$, where $p \leq q$, they found values for $\gamma(G)$, $\gamma_R(G)$, $\gamma_r(G)$ and $\gamma_s(G)$ proving that

$$\gamma_r(G) = \begin{cases} 2, \; p = 1, 2, q > 1 \\ 3, \; p = 3 \\ 4, \; p \leq 4. \end{cases}$$

Finally, they proposed all four domination numbers for a complete multipartite graph $K_{p_1, p_2, \ldots, p_t}$ where $p_1 \leq p_2 \leq \ldots \leq p_t$ and $t \geq 3$ (proving that γ_r is equal to 2 when $p_1 = 1, 2$ and equal to 3 when $p_1 \geq 3$), paths P_n and cycles C_n (proving that $\gamma_r = \gamma_s = \lceil 3n/7 \rceil$), and their products $P_m \times P_k$ and $C_m \times C_k$ (proving that $\gamma_r(P_m \times P_k) \leq \gamma_s(P_m \times P_k) \leq \lceil mk/3 \rceil + 2$ and $\gamma_r(C_m \times C_k) \leq \gamma_s(C_m \times C_k) \leq \lceil mk/3 \rceil$).

Pushpam and Mai (2011) characterized the class of the trees and split graphs for which $\gamma_r(G) = \gamma(G)$, found γ_r value for caterpillar, $2 \times n$ grid graphs and a complete binary tree and proved that $\gamma_r(C_n) = \gamma_r(P_n)$.

The weak Roman domination number of a corona of any two graphs and generalized web graphs is established by Lai et al. (2011). For instance, they prove that for a corona of a graph G of n vertices and a complete graph K_m, $\gamma_r(G \circ K_m) = n$, while for two graphs G and H with n and m vertices and $H \neq K_m$, $\gamma_r(G \circ H) = 2n$. Further, for a helm graph H_n and a web graph W_n, they show that $\gamma_r(H_n) = \gamma_r(W_n) = n + 1$, while for a generalized web graph $W_{t,3}$, when $t \geq 3$, $\gamma_r(W_{t,3}) = t + 2$. Proving that $\gamma_r(W_{3,4}) = 7$ and $\gamma_r(W_{3,5}) = 9$, they also gave generalization of a weak Roman domination number for a generalized web graphs $\gamma_r(W_{3,n})$ when $n \geq 6$ by setting $n = 3a + b$, for $0 \leq b \leq 2$ and proving that $\gamma_r(W_{3,n}) = 4a + 2$ when $b = 0$ and $\gamma_r(W_{3,n}) = 4a + 3 + b$ otherwise.

The impact on the weak Roman domination number of a simple connected graph G with connectivity of one, when one of its cut points is removed, is, for example, considered by Song et al. (2013).

The weak Roman domination number for $2 \times n$ grid graphs is determined by Song et al. (2011).

Nonetheless, some relations between several different domination numbers were nicely summarized by Chellali et al. (2014) who also showed that for all graphs, the weak Roman domination number is bounded above by the 2-rainbow domination number $\gamma_{r2}(G)$.

Motivated by Chellali et al. (2014), Alvarez-Ruiz et al. (2015) provided a constructive characterization of the trees for which the Roman domination number strongly equals to the weak Roman domination number. With the characterization based on five simple extension operations, Alvarado et al. (2015b) also revealed several structural properties of these trees.

Later, Alvarado et al. (2015a) proved that $\gamma_{r2}(G) \leq 2\gamma_r(G)$ for every graph G. Then, for a complete graph K_n they characterized the extremal graphs for this inequality that are $\{K_4, K_4 - e\}$-free, and showed that the recognition of the K_5-free extremal graphs is NP-hard (by the $K_n - e$ it was meant on a graph that arises by removing one edge from K_n).

Recently, Pushpam and Kamalam (2015a) found an efficient weak Roman domination number of some of the Myscielski graphs.

The idea of efficiency to the weak roman domination was extended by Pushpam et al, who also characterized certain classes of graphs that are efficiently weak Roman dominant (Pushpam and Kamalam 2015b).

In general case, it was proved that the RDP is NP-complete (i.e., Dreyer 2000; Shang and Hu 2007; Klobučar and Puljić 2014), but for some special classes of graphs, such as interval graphs, intersection graphs, co-graphs and distance-hereditary graphs, it can be solved in a linear time, see Klobučar and Puljić (2014). Given that the WRDP is a generalization of the RDP, the WRDP can be also considered as NP-complete problem. Nonetheless, Henning and Hedetniemi (2003) proved that WRDP is NP-complete, even restricted to bipartite and chordal graphs.

The linear time algorithm for computing the weak Roman domination number of a block graph is given by Liu et al. (2010).

Considering the algorithms for solving the WRDP, Chapelle et al. (2013) broke the trivial enumeration barrier of $O^*(3^n)$ (the notation $O^*(f(n))$ suppresses polynomial factors) providing two faster algorithms. Proving that the WRDP can be solved in $O^*(2^n)$ time needing exponential space, they described an $O^*(2.2279^n)$ algorithm using polynomial space. Their proposed results rely on structural properties of a WRDF, as well as on the best polynomial space algorithm for the Red-Blue Dominating Set problem.

Burger et al. Burger et al. (2013) gave relation between $\gamma(G)$, $\gamma_R(G)$ and $\gamma_r(G)$ and proposed the only one integer linear programming (ILP) formulation for the weak Roman domination problem to the knowledge of the author.

In this paper, a new ILP formulation will be proposed, proved to be correct and compared with the existing one. Thus, the first binary programing formulation of the WRDP will be reviewed here below.

For a graph $G = (V, E)$ represented by a vertex set V, $n = |V|$, and a set of edges E, $m = |E|$, let the adjacency matrix be denoted by a_{ij} for all $i \neq j$, with the convention that $a_{ii} = 1$ for all $i = 1, \dots, n$. Further, let X represents a set of vertices containing exactly one legion and let Y represents a set of vertices containing exactly two legions within it. Binary decision variables will be defined with

$$x_i = \begin{cases} 1, & i \in X \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$y_i = \begin{cases} 1, & i \in Y \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

for all $i = 1, \dots, n$ and

$$z_{ij} = \begin{cases} 1, & i\ (x_i = 0 \text{ and } y_i = 0) \text{ and } j\ (x_j = 1 \text{ or } y_j = 1) \\ & \text{form a swap set} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

for all $i, j = 1, \dots, n$, $i \neq j$ and $i, j \in V$.

Burger et. al. formulation of the weak Roman domination problem is:

$$\min \sum_{k=1}^{n} (x_k + 2y_k) \tag{4}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij}(x_j + y_j) \geq 1, \quad i = 1, \dots, n \tag{5}$$

$$x_k + y_k \leq 1, \quad k = 1, \dots, n \tag{6}$$

$$x_i + y_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ij} \geq 1 \quad i = 1, \dots, n \tag{7}$$

$$a_{ij}(x_j + y_j - x_i - y_i) \geq 2z_{ij}, \quad i, j = 1, \dots, n, j \neq i \tag{8}$$

$$y_j a_{kj} + a_{ki} + \sum_{\substack{l=1 \\ l \neq i \\ l \neq j}}^{n} a_{kl}(x_l + y_l) \geq z_{ij},$$
$$i, j, k = 1, \dots, n, i \neq j \tag{9}$$

$$x_i, y_i, z_{ij} \in \{0, 1\}, \quad i, j = 1, \dots, n. \tag{10}$$

The objective function value, given by (4), represents the weak Roman domination number $\gamma_r(G)$. Constraints (5) ensure that each vertex $i \in V$ with property $f(i) = 0$ is adjacent to at least one vertex j with property $f(j) > 0$. By the constraints (6), it is ensured that sets X and Y are mutually disjunctive. Further, by the constraints (7) it is ensured that for each unoccupied vertex i ($x_i = y_i = 0$) there exists a vertex j, $j \in N_i$ with whom it can form a swap, while by the constraints (8) it is ensured that each swap from j to i is valid. Furthermore, safety of every province after any single swap performed from j to i is preserved by the constraints (9). Constraints (10) deal with binary nature of decision variables x_i, y_i and z_{ij}.

Presented formulation has $n^2 + 2n$ variables, has $n^3 + n^2 + 3n$ constraints and will be referred as the $\mathcal{Old}$.
Note that in Burger et al. (2013), in the constraints (9) originally stand

$$y_j a_{ki} + a_{kj} + \sum_{\substack{l=1 \\ l \neq i \\ l \neq j}}^{n} a_{kl}(x_l + y_l) \geq z_{ij}.$$

We assume that it was a typing error since in that case mathematical formulation does not correspond to the WRDP.

3 Improved ILP formulation for the weak Roman domination problem

Inspired by the $\mathcal{Old}$ formulation, it was noticed that there is no need to calculate swap z_{ij} for each pair of vertices i and j. Instead of that, it is sufficient to calculate swap only between two adjacent vertices. Therefore, let $E' = \{(j, i) | (i, j) \in E\}$ and

$$z_e = \begin{cases} 1, & e = (i, j), \\ & i \ (x_i = 0 \text{ and } y_i = 0) \text{ and } j \ (x_j = 1 \text{ or } y_j = 1) \\ & \text{form a swap set} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

for all $e \in E \cup E'$ and, again, let binary decision variables x_i and y_i be defined with (1) and (2).

Now, new ILP formulation for the WRDP, on a graph $G = (V, E)$, can be described as follows:

$$\min \sum_{k \in V} (x_k + 2y_k) \tag{12}$$

subject to the constraints

$$x_i + y_i + \sum_{e \ni i} z_e \geq 1, \quad i \in V \tag{13}$$

$$x_j + y_j - x_i - y_i + 1 \geq 2z_e, \quad e = (i, j) \in E \cup E' \tag{14}$$

$$y_j + \sum_{\substack{l \in N_k \\ l \neq i, \\ l \neq j}} (x_l + y_l) \geq z_e,$$
$$e = (i, j) \in E \cup E', k \in N_j \setminus N_i \tag{15}$$

$$x_i, y_i, z_e \in \{0, 1\} \quad i \in V, e \in E \cup E'. \tag{16}$$

Again, the objective function value, given by (12), represents the weak Roman domination number $\gamma_r(G)$. Constraints (13) ensure that vertex i is defended, or it is adjacent to a defended vertex with whom it can form a swap. Validity of the swap from the vertex j to the vertex i is ensured by the constraints (14). Also, by the constraints (15), safety of the Empire is preserved after any single swap performed from j to i. Binary nature of decision variables is ensured by (16).

New ILP formulation will be referred as the $\mathcal{New}$ and has $2n + m$ variables and $3n + 2m$ constraints.

Note that, for a complete planar graph with n vertices, the $\mathcal{New}$ formulation consists of $\frac{1}{2}n^2 + \frac{3}{2}n$ variables and $n^2 + 2n$ constraints. Therefore, when the $\mathcal{New}$ formulation is used, the number of the variables is reduced by at least $\frac{1}{2}(n^2 + n)$ while the number of the constraints is reduced by $n^3 + n$.

The following theorem proves that $\mathcal{New}$ formulation is equivalent to the $\mathcal{Old}$ formulation.

Theorem 1 *For every simple graph G, optimal objective function value of the $\mathcal{Old}$ formulation (4)–(10) is equal to the optimal objective function value of the $\mathcal{New}$ formulation (12)–(16).*

Proof (⇒) Let a feasible solution to the $\mathcal{Old}$ formulation be represented by a set (X', Y', Z'), $X' = (x'_1, \dots, x'_n)$, $Y' = (y'_1, \dots, y'_n)$, $Z' = (z'_{11}, \dots, z_{nn})$, $n = |V|$, and similarly, let a set (X'', Y'', Z'') of variables x''_i, y''_i and z''_e be defined such that $x''_i = x'_i$, $y''_i = y'_i$, $i = 1, \ldots, n$ and

$$z''_e = \begin{cases} 1, & e = (i,j), z'_{ij} = 1 \\ 0, & e = (i,j), z'_{ij} = 0 \end{cases}, \quad e \in E \cup E'. \tag{17}$$

Note that variables z'_{ij} are defined for each pair of vertices (i, j), while z''_e are defined only for edges that exist in a graph G. Hence, z''_e are defined only when $a_{ij} = 1, e = (i, j) \in E \cup E'$. Also note that, by the conditions (8), for $a_{ij} = 0$ it follows that $z'_{ij} = 0$. Thus, variables z''_e are well defined since in the conditions (7) $z'_{ij} = 0$ are part of the sum, and given that zero is neutral for summing, they do not affect the given sum. Even more, for those z'_{ij}, conditions (8) and (9) become trivial. From the above consideration, for each $i \in V$, it follows that $\sum_{j \in V} z'_{ij}$ is equal to the $\sum_{e \ni i} z''_e$. Therefore, conditions (13) follow from the conditions (7). Also note that conditions (8) are nontrivial only when $a_{ij} = 1$. Given that, and from the definition of the z''_e, directly follows that conditions (7) imply (14).

In order to prove conditions (15), it is necessary to consider next three cases:

Case 1 $k \notin N_j$. Since k and j are not in the neighborhood, it follows that $a_{kj} = 0$, which means that left-hand side (LHS) of the conditions (9),

$$y'_j a_{kj} + a_{ki} + \sum_{l=1}^{n} a_{kl}(x'_l + y'_l) - a_{ki}(x'_i + y'_i) - a_{kj}(x'_j + y'_j)$$

becomes

$$-a_{kj}x'_j + (1 - x'_i - y'_i)a_{ki} + \sum_{l=1}^{n} a_{kl}(x'_l + y'_l).$$

Further, $a_{kj}x'_j = 0$ holds from the starting assumption, while from the conditions (6) it follows that $1 - x'_i - y'_i \geq 0$. Finally, from the conditions (5) it follows that

$$- a_{kj}x'_j + (1 - x'_i - y'_i)a_{ki} + \sum_{l=1}^{n} a_{kl}(x'_l + y'_l) \geq \sum_{l=1}^{n} a_{kl}(x'_l + y'_l) \geq 1 \geq z'_{ij}.$$

From the above consideration, it is obvious that in *Case 1* conditions (9) are redundant.

Case 2 $k \in N_i \cap N_j$. By the assumption that the vertex k is in the neighborhood of the vertices i and j, it follows that $a_{ki} = a_{kj} = 1$. Since $a_{kj}y'_j \geq 0$ and $\sum_{\substack{l=1 \\ l \neq i \\ l \neq j}} a_{kl}(x'_l + y'_l) \geq 0$, it follows that LHS of the conditions (9) is greater or equal to $a_{ki} = 1$, implying that LHS is greater or equal to z'_{ij}.

From the above consideration, it is obvious that in *Case 2* conditions (9) are redundant.

Case 3 $k \in N_j \setminus N_i$. Assuming that vertex k is in the neighborhood of the j and not in the neighborhood of the i, it follows that $a_{kj} = 1$ and $a_{ki} = 0$. Therefore, conditions (9) becomes $y'_j + \sum_{\substack{l=1 \\ l \neq i \\ l \neq j}} a_{kl}(x'_l + y'_l) \geq z'_{ij}$. By the definition of the variables z''_e, it follows that conditions (15) hold.

Since that *Case 1* and *Case 2* are redundant and do not have influence on the feasible solution space of the model (4)–(10), it follows that they can be ignored.

Variables x''_i, y''_i and z''_e are binary by the definition, i.e., conditions (16) hold also.

Finally, since $x''_k = x'_k$ and $y''_k = y'_k$, it follows that objective function values are equal, i.e.,

$$Obj_{\mathcal{Old}} = \sum_{k \in V} (x'_k + 2y'_k) = \sum_{k \in V} (x''_k + 2y''_k) = Obj_{\mathcal{New}}.$$

Even more, objective function value of the $\mathcal{Old}$ formulation is greater or equal to the objective function value of the $\mathcal{New}$ formulation, since feasible solution of the $\mathcal{Old}$ formulation is feasible solution of the $\mathcal{New}$ formulation.

($\Leftarrow$) Let a set (X'', Y'', Z'') represents a feasible solution to the $\mathcal{New}$ formulation $(X'' = (x''_1, \ldots, x''_n)$, $Y'' = (y''_1, \ldots, y''_n)$, $Z'' = (z''_1, \ldots, z''_m))$, $n = |V|$, $m = |E|$ and let a set (X', Y', Z') of variables x'_i, y'_i and z'_{ij}, $i, j = 1, \ldots, n$ be defined such that $y'_i = y''_i$, $i = 1, \ldots, n$,

$$x'_i = \begin{cases} x''_i, & x''_i + y''_i \leq 1 \\ x''_i - 1, & x''_i + y''_i = 2 \end{cases}$$

and

$$z'_{ij} = \begin{cases} z''_e, & x''_j + y''_j \leq 1, e = (i,j) \in E \cup E' \\ 0, & \text{otherwise.} \end{cases}$$

In order to prove conditions (5)–(9), two cases will be observed:

(*1*) $x''_i + y''_i \leq 1$ and
(*2*) $x''_i + y''_i = 2$.

Conditions (6) trivially hold since, in (*1*.), $x'_k + y'_k = x''_k + y''_k \leq 1$ and, in (*2*.), $x'_k + y'_k = x''_k - 1 + y''_k = 1 \leq 1$.

Starting from the conditions (13) and by the definition of the variables z'_{ij}, conditions (7) hold also:

$$x'_i + y'_i + \sum_{\substack{j=1\\ j\neq i}}^{n} z'_{ij} =$$

$$\text{in (1.),}\quad x''_i + y''_i + \sum_{\substack{e=(i,j)\in E\cup E'\\ x''_j+y''_j\le 1}} z'_{ij} + \sum_{\substack{e=(i,j)\in E\cup E'\\ x''_j+y''_j=2}} z'_{ij} + \sum_{e=(i,j)\notin E\cup E'} z'_{ij} =$$

$$x''_i + y''_i + \sum_{\substack{e=(i,j)\in E\cup E'\\ x''_j+y''_j\le 1}} z'_{ij} =$$

$$x''_i + y''_i + \sum_{e\ni i} z''_e - \sum_{\substack{e=(j,i)\in E\cup E'\\ x''_j+y''_j=2}} z''_e =$$

$$x''_i + y''_i + \sum_{e\ni i} z''_e \ge 1$$

$$\text{in (2.),}\quad x''_i - 1 + y''_i + \sum_{\substack{j=1\\ j\neq i}}^{n} z'_{ij} \ge x''_i - 1 + y''_i = 1$$

for all $i = 1, \ldots, n$.
Above formulas are correct since $\sum_{\substack{e=(i,j)\in E\cup E'\\ x''_j+y''_j=2}} z'_{ij} = 0$ and $\sum_{e=(i,j)\notin E\cup E'} z'_{ij} = 0$ by the definition of the variable z'_{ij}, while $\sum_{\substack{e=(j,i)\in E\cup E'\\ x''_j+y''_j=2}} z''_e = 0$ because of the constraint (14) which in case $e = (j, i)$ should read as follows $x''_i + y''_i - x''_j - y''_j + 1 \ge 2z''_e$ directly implying that $z''_e = 0$.

Validity of the conditions (8) can be shown also: For $a_{ij} = 0$, conditions (8) trivially hold since $0 \ge 0$, while for $a_{ij} = 1$ four cases are considered:

– (*1.*) holds for all $i, j = 1, \ldots, n$.

$$x'_j+y'_j-(x'_i+y'_i)+1 = x''_j+y''_j-(x''_i+y''_i)+1 \ge 2z''_e = 2z'_{ij}.$$

– (*1.*) holds for i while (*2.*) holds for j, $i, j = 1, \ldots, n$.

$$x'_j + y'_j - (x'_i + y'_i) + 1 = x''_j - 1 + y''_j - (x''_i + y''_i) + 1$$
$$= x''_j + y''_j - (x''_i + y''_i) = 2 - (x''_i + y''_i) \ge 1 \ge 0 = 2z'_{ij}$$

since, by the definition, $z'_{ij} = 0$ when $x''_j + y''_j = 2$.

– (*1.*) holds for j while (*2.*) holds for i, $i, j = 1, \ldots, n$.

$$x'_j + y'_j - (x'_i + y'_i) + 1 = x''_j + y''_j - (x''_i - 1 + y''_i) + 1$$
$$= x''_j + y''_j - (x''_i + y''_i) + 2 \ge 2z''_e \ge 2z'_{ij}.$$

– (*2.*) holds for all $i, j = 1, \ldots, n$.

$$x'_j + y'_j - (x'_i + y'_i) + 1$$
$$= x''_j - 1 + y''_j - (x''_i - 1 + y''_i) + 1 =$$
$$x''_j + y''_j - (x''_i + y''_i) + 1 \ge 2z''_e \ge 2z'_{ij}.$$

Combining all four cases together with the case when $a_{ij} = 0$, it follows that conditions (8) are satisfied.

Conditions (5) which can be equally written as

$$x'_i + y'_i + \sum_{j\in N_i} (x'_j + y'_j) \ge 1$$

are satisfied for (*2.*) since

$$x'_i + y'_i + \sum_{j\in N_i} (x'_j + y'_j)$$
$$= x''_i - 1 + y''_i + \sum_{j\in N_i} (x'_j + y'_j) \ge 1 + \sum_{j\in N_i} (x'_j + y'_j) \ge 1.$$

If (*1.*) holds for i and conditions $x''_j + y''_j \le 1$ hold for all j, conditions (5) can be written as $x''_i + y''_i + \sum_{j\in N_i} (x''_j + y''_j) \ge 1$. The last relation holds if at least one of next two relations is satisfied: $x''_i + y''_i \ge 1$ or $\sum_{j\in N_i} (x''_j + y''_j) \ge 1$. Assuming that conditions (13)–(16) are satisfied, from the conditions (13) it follows that at least one of the relations $x''_i + y''_i \ge 1$ and $\sum_{e\ni i} z''_e \ge 1$ holds. If the first relation holds, conditions (5) are satisfied. If the second relation holds, then there exists an edge $\bar{e}$ such that $z''_{\bar{e}} = 1$. Now, from the condition (14) it follows that $x''_j + y''_j \ge 1$ for $\bar{e} = (i, j)$, proving that conditions (5) are satisfied again.

If (*1.*) holds for i and there exists $\bar{j}$ such that $x''_{\bar{j}} + y''_{\bar{j}} = 2$, conditions (5) can be written as $x'_i + y'_i + \sum_{j\in N_i, j\neq \bar{j}} (x'_j + y'_j) + (x'_{\bar{j}} + y'_{\bar{j}}) \ge x'_{\bar{j}} + y'_{\bar{j}} = x''_{\bar{j}} - 1 + y''_{\bar{j}} = 1$.

In order to show validity of the conditions (9), next three cases should be considered.

1. $a_{kj} = a_{ki} = 0$.
   Keeping in mind that constraints (5) hold, constraints (9) can be equally written as

$$a_{kj} y'_j + a_{ki} + \sum_{\substack{l=1\\ l\neq i\\ l\neq j}}^{n} a_{kl}(x'_l + y'_l)$$
$$= \sum_{l=1}^{n} a_{kl}(x'_l + y'_l) - a_{ki}(x'_i + y'_i) - a_{kj}(x'_j + y'_j)$$
$$= \sum_{l=1}^{n} a_{kl}(x'_l + y'_l) \ge 1 \ge z'_{ij}$$

   from which it can be concluded that they are satisfied.

2. $a_{kj} = 1, a_{ki} = 0$.
   Keeping in mind that constraints (15) are also satisfied, constraints (9) are satisfied again,

$$a_{kj}y'_j + a_{ki} + \sum_{\substack{l=1\\ l\neq i\\ l\neq j}}^{n} a_{kl}(x'_l + y'_l) = y'_j + \sum_{\substack{l\in N_k\\ l\neq i\\ l\neq j}} (x'_l + y'_l) =$$

   case $((\forall l)(x''_l + y''_l \leq 1))$:

$$y''_j + \sum_{\substack{l\in N_k\\ l\neq i\\ l\neq j}} (x''_l + y''_l) \geq z''_e \geq z'_{ij}$$

   case $((\exists \bar{l})(x''_{\bar{l}} + y''_{\bar{l}} = 2))$:

$$y'_j + \sum_{\substack{l\in N_k\\ l\neq i\\ l\neq j\\ l\neq \bar{l}}} (x'_l + y'_l) + (x'_{\bar{l}} + y'_{\bar{l}}) \geq x'_{\bar{l}} + y'_{\bar{l}} = 1 \geq z'_{ij}$$

3. $a_{kj} = 0 \wedge a_{ki} = 1$ or $a_{kj} = a_{ki} = 1$.
   Since $a_{kj}y'_i \geq 0$ and $a_{kl}(x'_l + y'_l) \geq 0$ for all $k, l = 1, \ldots, n, l \neq i$, constraints (9) are again satisfied,

$$a_{kj}y'_j + a_{ki} + \sum_{\substack{l=1\\ l\neq i\\ l\neq j}}^{n} a_{kl}(x'_l + y'_l) =$$

$$a_{kj}y'_i + 1 + \sum_{\substack{l=1\\ l\neq i\\ l\neq j}}^{n} a_{kl}(x'_l + y'_l) \geq 1 \geq z'_{ij}.$$

From the above, it can be concluded that constraints (9) hold.

Finally, since $x'_k \leq x''_k$ and $y'_k = y''_k$, it follows that objective function value of the $\mathcal{Old}$ formulation is less or equal to the objective function value of the $\mathcal{New}$ formulation, i.e., $Obj_{\mathcal{Old}} \leq Obj_{\mathcal{New}}$. Even more, feasible solution to the $\mathcal{New}$ formulation is a feasible solution to the $\mathcal{Old}$ formulation.

Combining given inequalities theorem is proved, i.e., $Obj_{\mathcal{New}} = Obj_{\mathcal{Old}}$. □

4 Computational results

In this section, computational results which show effectiveness of the proposed ILP formulation method are summarized. Direct comparison is given only between the proposed ILP formulation and mathematical formulation presented in Burger et al. (2013). Comparison with algorithms presented in Liu et al. (2010) and Chapelle et al. (2013) is not possible since both of these two papers are theoretical and without numerical results. Tests were run using CPLEX 12.6 and Gurobi 5.6 optimization solvers and carried out on Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz 2.39GHz, with 8GB RAM under Windows 8.1 operating system.

For all computational experiments grid, planar, net and randomly generated graphs were used. Grid, planar and net graphs are well-known type of graphs, while randomly generated graphs are provided by Vincenzo Currò and were specially generated for his Ph.D. theses Currò (2014). The set of grid graph instances consists of 42 instances where the smallest one has 40 vertices, while the greatest one has 100 vertices. The set of planar graph instances consists only of 6 instances with 10, 20, 30, 50, 100 and 150 vertices. The set of net graphs consists only of four instances with 100, 200, 400 and 600 vertices, while randomly generated graphs consists of 32 instances from which 18 of them are with 50 and 14 of them are with 100 vertices.

All instances were tested using both optimization solvers. The results are shown for each type of graph instance in separate tables. For all tables, in the first three columns the name of the instance, the number of vertices and the number of edges are given. In case that optimizations solvers succeeded in finding and proving the optimal solution to the tested instances, the objective function value is shown at the fourth column (*val*). For each ILP formulation, the results (objective function value and optimization solvers' execution time) are shown in the next two sets of four columns such that the results of the $\mathcal{Old}$ formulation are followed by the results of the $\mathcal{New}$ formulation. For instances where optimal solution was not reached within the time limit of 7200 seconds, the sign "*" is shown. Also for instances where optimization solver stopped because of "out of memory status," the sign "–" stands.

Comparing the execution time of two solvers and two formulations, it is obvious that both solvers were more successful by using $\mathcal{New}$ formulation.

From Table 1, based on the $\mathcal{Old}$ formulation, CPLEX reached optimal solution value for 22 instances and Gurobi for 9 instances, while based on the $\mathcal{New}$ formulation, CPLEX and Gurobi were successful in finding optimal solution for 32 instances. Actually, based on the $\mathcal{New}$ formulation, Gurobi succeeds in finding the solution value for all grid graph instances, but because of the running time limit, not all found solution values are verified as optimal. Even more, from Table 1, for all instances where solution is found and verified as optimal, Gurobi running time was less when $\mathcal{New}$ formulation is used. For instances, with at most 60 vertices CPLEX, based on the both formulations, was successful in finding optimal solution values with very small differences in run-

Table 1 Computational results on grid graph instances

Instance				*Old* formulation				*New* formulation			
	$\|V\|$	$\|E\|$	opt	val.	t_{gur}	value	t_{cpl}	val	t_{gur}	val	t_{cpl}
grid04 × 10	40	66	15	15	497.37	15	4.954	15	9.16	15	4.109
grid05 × 08	40	67	14	14	551.75	14	5.191	14	7.56	14	4.64
grid03 × 14	42	67	16	16	314.72	16	4.829	16	13.64	16	5.372
grid06 × 07	42	71	15	15	454.75	15	6.927	15	12.28	15	5.801
grid04 × 11	44	73	16	16	912.76	16	6.13	16	14.1	16	5.5
grid03 × 15	45	72	17	17	712.63	17	8.36	17	19.69	17	7.789
grid05 × 09	45	76	16	16	942.68	16	7.908	16	19.48	16	10.781
grid04 × 12	48	80	17	17	891.93	17	14.364	17	14.91	17	12.84
grid06 × 08	48	82	18	18	*	18	26.662	18	108.8	18	25.499
grid07 × 07	49	84	18	18	*	18	9.845	18	80.37	18	12.844
grid05 × 10	50	85	18	18	5953.67	18	11.469	18	71.83	18	10.61
grid04 × 13	52	87	19	22	–	19	13.16	19	69.13	19	11.813
grid06 × 09	54	93	19	55	–	19	26.988	19	79.64	19	25.539
grid05 × 11	55	94	19	59	–	19	14.365	19	107.34	19	11.424
grid04 × 14	56	94	20	25	–	20	35.6	20	52.53	20	35.326
grid07 × 08	56	97	20	23	–	20	21.882	20	107.63	20	42.48
grid04 × 15	60	101	22	27	–	22	40.256	22	172.62	22	51.518
grid05 × 12	60	103	21	28	–	21	41.364	21	127.57	21	14.88
grid06 × 10	60	104	21	26	–	21	51.458	21	58.41	21	35.713
grid07 × 09	63	110	22	27	–	22	995.68	22	101.89	22	70.259
grid08 × 08	64	112	23	26	–	23	*	23	1664.53	23	171.925
grid05 × 13	65	112	23	28	–	23	1825	23	195.49	23	67.007
grid06 × 11	66	115	24	29	–	24	*	24	1001.9	24	381.771
grid05 × 14	70	121	24	28	–	24	5368.9	24	677.63	24	73.489
grid07 × 10	70	123	25	30	–	25	*	25	829.21	25	618.089
grid06 × 12	72	126	26	31	–	27	-	26	3254.34	26	1166.405
grid08 × 09	72	127	25	32	–	25	*	25	4195.49	25	435.146
grid05 × 15	75	130	26	31	–	26	*	26	288.06	26	465.006
grid07 × 11	77	136	27	32	–	28	*	27	1543.24	27	988.596
grid06 × 13	78	137	27	33	–	28	*	27	6657.33	27	1005.126
grid08 × 10	80	142	28	33	–	28	*	28	3693.43	28	2162.812
grid09 × 09	81	144	28	35	–	29	*	28	*	28	737.579
grid06 × 14	84	148		34	–	30	*	30	*	30	–
grid07 × 12	84	149	29	35	–	30	*	29	4637.38	30	–
grid08 × 11	88	157		36	–	31	*	31	*	31	–
grid06 × 15	90	159		95	–	32	*	32	*	33	–
grid09 × 10	90	161		90	–	32	*	31	*	32	–
grid07 × 13	91	162		109	–	32	*	32	*	33	–
grid08 × 12	96	172		106	–	38	–	33	*	34	–
grid07 × 14	98	175		115	–	34	*	35	*	34	–
grid09 × 11	99	178		110	–	35	*	35	*	35	–
grid10 × 10	100	180		100	–	36	–	35	*	36	–

ning time, but for instances such as grid07 × 09, grid05 × 13 and grid05 × 14, when *New* formulation is used, running time was significantly less. Also, for instances with more than 60 vertices, where CPLEX based on the *New* formulation was successful in finding optimal solution, running time was lesser than 2200 seconds, while CPLEX based on the *Old* formulation, for the same instances, was either stopped because of "out of memory" status, either because of the time limit.

Table 2 Computational results on net graph instances

Instance				*Old* formulation				*New* formulation			
	$\lvert V\rvert$	$\lvert E\rvert$	opt	val.	t_{gur}	value	t_{cpl}	val	t_{gur}	val	t_{cpl}
net-10-10	100	342	20	90	–	21	–	20	598.98	20	148.213
net-10-20	200	712			–		–	42	*	40	–
net-20-20	400	1482			–		–	83	*	87	–
net-30-20	600	2252			–		–	122	*	142	–

Table 3 Computational results on planar graph instances

Instance				*Old* formulation				*New* formulation			
	$\lvert V\rvert$	$\lvert E\rvert$	opt	val.	t_{gur}	value	t_{cpl}	val	t_{gur}	val	t_{cpl}
plan10	10	27	3	3	0.31	3	0.156	3	0.17	3	0.206
plan20	20	105	3	3	2.75	3	1.363	3	1.36	3	1.423
plan30	30	182	5	5	26.73	5	8.724	5	11.45	5	7.49
plan50	50	465	6	6	–	6	302.82	6	98.49	6	153.215
plan100	100	1540			–		–	10	*	9	–
plan150	150	2867			–		–	13	*		–

From Table 2, it can be seen that optimal solution value for net graph instances was reached only for one instance, Net-10-10. Based on the *New* formulation, both CPLEX and Gurobi optimization solvers were stopped either because of "out of memory" status, either because of the time limitation but still were able to provide some solution values for other three instances. Based on the *Old* formulation, both optimization solvers were unsuccessful in providing any solution value for these three instances.

Table 3 consists of computational results on the planar graph instances. Optimal solution value was reached for instances with at most 50 vertices by using both optimization solvers based on the both formulations and mostly with lesser running time when the *New* formulation is used. Both optimization solvers based on the *Old* formulation were unsuccessful in solving instances plan100 and plan150 because of "out of memory" status, while Gurobi optimization solver based on the *New* formulation provides some solution value within the time limit of 2 hours.

Finally, from Table 4, computational results on randomly generated graphs are given. Usage of the *Old* formulation provides optimal solution value for 24 instances with CPLEX and 13 instances with Gurobi optimization solver, while usage of the *New* formulation provides optimal solution value for 24 instances when CPLEX is used and for 25 when Gurobi optimization solver is used. Further, for instances when optimal solution value was reached by using both formulations, at almost all instances, solvers' running times are lesser when *New* formulation is used. Moreover, on instances where provided solution value was not verified as optimal (Random-100-8, Random-100-9, Random-100-10 and Random-100-20), better solution values are provided when *New* formulation is used. Furthermore, on instances Random-100-30, Random-100-40 and Random-100-50, both optimization solvers, based on the *Old* formulation, were unsuccessful in providing any solution value before status "out of memory" occurs.

5 Conclusions

In this paper, the weak Roman domination problem is considered. New, improved ILP formulation is proposed, proved to be correct and compared with the one known from the literature. It was shown that set of the constraints (7) can be excluded, and that in some cases the set of the constraints (9), of the known formulation, are redundant. Even more, it was shown that it is sufficient to calculate swap only between two adjacent vertices. Since proposed formulation uses lesser number of constraints and lesser number of variables than the one known from the literature, significant improvements in the computational effort for solving *New* formulation were expected. Therefore, computational experiments were carried out on a grid, planar, net and randomly generated graphs up to 600 vertices from the literature. It was shown that CPLEX and Gurobi solvers, based on the *New* formulation, provide optimal solutions for a larger number of instances; then, it was the case with optimization solvers based on the *Old* formulation. Even more, running times are mostly lesser when *New* formulation is used. Regardless the fact that running times using CPLEX were lesser than the running times

Table 4 Computational results on randomly generated graph instances

Instance				*Old* formulation				*New* formulation			
	$\|V\|$	$\|E\|$	opt	val.	t_{gur}	value	t_{cpl}	val	t_{gur}	val	t_{cpl}
Random-50-1	50	49	24	28	–	24	1.125	24	0.72	24	0.281
Random-50-2	50	49	23	23	85.66	23	1.156	23	1.48	23	0.343
Random-50-3	50	58	24	24	53.95	24	1.265	24	0.66	24	0.39
Random-50-4	50	54	24	24	66.97	24	1.239	24	0.58	24	0.484
Random-50-5	50	67	22	22	79.59	22	1.635	22	1.41	22	0.968
Random-50-6	50	86	19	19	118.34	19	4.915	19	12.31	19	2.053
Random-50-7	50	84	19	19	248.14	19	4.677	19	9.33	19	3.171
Random-50-8	50	95	17	17	226.28	17	15.308	17	4.11	17	3.093
Random-50-9	50	108	17	17	865.54	17	90.188	17	39.31	17	26.373
Random-50-10	50	112	16	16	716.16	16	43.673	16	34.2	16	6.781
Random-50-20	50	248	9	13	–	9	1304.7	9	792.33	9	346.264
Random-50-30	50	373	7	9	–	7	1143.9	7	943.77	7	476.278
Random-50-40	50	475	6	9	–	6	1767.9	6	3734.27	6	1447.318
Random-50-50	50	597	5	6	–	5	1856.4	5	1545.06	5	1779.272
Random-50-60	50	739	4	6	–	4	509.08	4	210.71	4	260.999
Random-50-70	50	860	3	5	–	3	203.18	3	156.14	3	168.483
Random-50-80	50	980	3	3	381.44	3	90.813	3	172.01	3	235.731
Random-50-90	50	1103	2	2	248.67	2	120.87	2	36.53	2	38.206
Random-100-1	100	100	46	123	–	46	8.769	46	2.61	46	0.64
Random-100-2	100	109	46	125	–	46	11.385	46	5.3	46	0.843
Random-100-3	100	181	37	109	–	37	704.88	37	23.17	37	7.421
Random-100-4	100	206	34	99	–	34	2533.2	34	66.7	34	61.702
Random-100-5	100	231	32	101	–	35	–	32	562.64	32	164.502
Random-100-6	100	321	26	90	–	28	*	26	5806.74	26	–
Random-100-7	100	317	25	87	–	26	*	25	4493.7	25	4009.377
Random-100-8	100	317		91	–	25	*	23	*	24	*
Random-100-9	100	430		81	–	23	*	21	*	22	*
Random-100-10	100	498		76	–	22	–	19	*	20	*
Random-100-20	100	981		64	–		–	12	*	12	*
Random-100-30	100	1477			–		–	11	–	11	–
Random-100-40	100	1945			–		–	9	–		–
Random-100-50	100	2483			–		–	7	–		–

using Gurobi optimization solver, usage of Gurobi optimization solver provides solution values for some instances even when CPLEX was stopped because of the memory limitations.

Given that optimization solvers were unsuccessful in providing solution values for larger graph instances than the one with 100 vertices, designing an exact method or even metaheuristic method for solving the proposed mathematical formulation is matter of the future work.

Compliance with ethical standards

Conflict of interest The author declares that there is no conflict of interest regarding the publication of this paper.

References

Ahangar HA, Henning MA, Löwenstein C, Zhao Y, Samodivkin V (2014) Signed roman domination in graphs. J Comb Optim 27(2):241–255

Alvarado JD, Dantas S, Rautenbach D (2015a) Relating 2-rainbow domination to weak roman domination. arXiv preprint arXiv:1512.01067

Alvarado JD, Dantas S, Rautenbach D (2015b) Strong Equality of Roman and Weak Roman Domination in Trees. arXiv preprint arXiv:1507.04901

Alvarez-Ruiz M, Yero IG, Mediavilla-Gradolph T, Valenzuela J (2015) A stronger vision for roman domination in graphs. arXiv preprint arXiv:1502.03933

Burger A, De Villiers A, Van Vuuren J (2013) A binary programming approach towards achieving effective graph protection. In: Proceedings of the 2013 ORSSA annual conference, ORSSA, 2013, pp 19–30

Chang GJ, Chen SH, Liu CH (2014) Edge roman domination on graphs. arXiv preprint arXiv:1405.5622

Chapelle M, Cochefert M, Couturier JF, Kratsch D, Liedloff M, Perez A (2013) Exact algorithms for weak roman domination. In: Combinatorial Algorithms, Springer, pp 81–93

Chellali M, Haynes TW, Hedetniemi ST (2014) Bounds on weak roman and 2-rainbow domination numbers. Discrete Appl Math 178:27–32

Cockayne E, Grobler P, Grundlingh W, Munganga J, Jv Vuuren (2005) Protection of a graph. Util Math 67:19–32

Currò V (2014) The roman domination problem on grid graphs. Ph.D. thesis, Università di Catania

Dreyer PA Jr (2000) Applications and variations of domination in graphs. Ph.D. thesis, Citeseer

Henning MA (2003) Defending the roman empire from multiple attacks. Discret Math 271(1):101–115

Henning MA, Hedetniemi ST (2003) Defending the roman empire a new strategy. Discret Math 266(1):239–251

Klobučar A, Puljić I (2014) Some results for roman domination number on cardinal product of paths and cycles. Kragujev J Math 38(1):83–94

Lai YL, Lin CT, Ho HM (2011) Weak roman domination on graphs. In: Proceedings of the 28th workshop on combinatorial mathematics and computation theory, Penghu University of Science and Technology, Penghu, Taiwan, May 27–28, 2011, pp 224–214

Liedloff M, Kloks T, Liu J, Peng SL (2005) Roman domination over some graph classes. In: Graph-Theoretic Concepts in Computer Science, Springer, pp 103–114

Liu CS, Peng SL, Tang CY (2010) Weak Roman Domination on Block Graphs. In: Proceedings of The 27th workshop on combinatorial mathematics and computation theory, Providence University, Taichung, Taiwan, April 30–May 1, 2010, pp 86–89

Pushpam PRL, Kamalam M (2015) Efficient weak roman domination in myscielski graphs. Int J Pure Eng Math (IJPEM) 3(2):93–100

Pushpam PRL, Kamalam M (2015) Efficient weak roman domination in graphs. Int J Pure Appl Math 101(5):701–710

Pushpam PRL, Mai TM (2011) Weak roman domination in graphs. Discuss Math Graph Theory 31(1):161–170

ReVelle CS, Rosing KE (2000) Defendens imperium romanum: a classical problem in military strategy. Am Math Mon 107(7):585–594

Shang W, Hu X (2007) The roman domination problem in unit disk graphs. In: Computational Science–ICCS 2007, Springer, pp 305–312

Song X, Yang J, Xie Y (2011) Weak Roman domination in 2xn grid graphs. J Henan Univ (Nat Sci) 41(1): 4–9 (**in Chinese**)

Song X, Bian J, Yin W (2013) Six safe grades on weak roman domination. J Henan Univ (Nat Sci) 5:002

Stewart I (1999) Defend the roman empire!. Sci Am 281:136–138

Targhi M, Rad NJ, Moradi MS (2012) Properties of independent roman domination in graphs. Australas J Combin 52:11–18

Computing and Informatics, Vol. 38, 2019, 57–84, doi: 10.31577/cai_2019_1_57

VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING ROMAN AND WEAK ROMAN DOMINATION PROBLEMS ON GRAPHS

Marija Ivanović

Faculty of Mathematics
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: marijai@math.rs

Dragan Urošević

Mathematical Institute, SANU
Kneza Mihaila 36
11 000 Belgrade, Serbia
e-mail: draganu@mi.sanu.ac.rs

Abstract. In this paper Roman and weak Roman domination problems on graphs are considered. Given that both problems are NP hard, a new heuristic approach, based on a Variable Neighborhood Search (VNS), is presented. The presented algorithm is tested on instances known from the literature, with up to 600 vertices. The VNS approach is justified since it was able to achieve an optimal solution value on the majority of instances where the optimal solution value is known. Also, for the majority of instances where optimization solvers found a solution value but were unable to prove it to be optimal, the VNS algorithm achieves an even better solution value.

Keywords: Roman domination in graphs, weak Roman domination in graphs, combinatorial optimization, metaheuristic, variable neighborhood search

Mathematics Subject Classification 2010: 05C69, 05C85, 90C10

1 INTRODUCTION

The Roman domination problem (RD problem) was introduced by ReVelle and Rosing [1] and Cockayne et al. [2] and can be interpreted as follows.

Assuming that any province of the Roman Empire is considered to be safe if there is at least one legion (of maximum 2) stationed within it, the RD problem requires that every unsafe province must be adjacent to a province with at least two legions stationed within it and the total number of stationed legions within all provinces of the Roman Empire is minimal.

In a graph terminology, let $G = (V, E)$ be a simple undirected graph with a vertex set V such that each vertex $u \in V$ represents a province of the Roman Empire and each edge, $e \in E$, represents an existing connection between two provinces. Let f be a function $f : V \to \{0, 1, 2\}$ and let the weight of the vertex u, denoted by $f(u)$, represent the number of legions stationed at province u. Further, let the weight of the function f be calculated by a formula $\sum_{v \in V} f(v)$. Function f is called a Roman dominating function (RD function) if every vertex u such that $f(u) = 0$ is adjacent to a vertex v such that $f(v) = 2$. The Roman domination problem is to find an RD function f of a graph G with the smallest weight. The smallest weight of the RD function f, denoted by $\gamma_R(G)$, is known as the Roman domination number.

We illustrate the Roman domination problem in the example below.

Example 1. Let us assume that the Roman Empire can be described by a graph $G = (V, E)$ as it is presented below, in Figure 1.

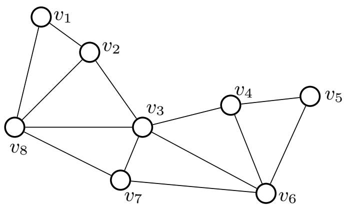


Figure 1: Graph $G = (V, E)$

The optimal number of legions necessary to defend the given graph is 4, provinces represented by vertices v_1 and v_5 are with one stationed legion, province represented by vertex v_3 is with two stationed legions and all other provinces are without stationed legions. With the given schedule, vertices v_1, v_3 and v_5 are defended because they have at least one legion stationed within it, while v_2, v_4, v_6, v_7 and v_8 are defended since they are in the neighborhood of the vertex v_3, which is with two stationed legions. The optimal solution to the proposed problem is illustrated in Figure 2, where vertices are marked by black squares if they are representing provinces with two stationed legions, marked by red circles if they are representing provinces with one stationed legion, and marked by white circles if they are representing provinces without stationed legions.

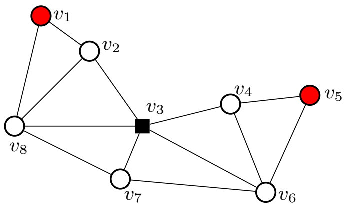


Figure 2: Illustrated solution of the RD problem on a graph G defined in the Example 1

In order to reduce the number of legions necessary to defend the Roman Empire against a single attack, Henning and Hedetniemi [3] introduced the weak Roman domination problem (WRD problem) as a variant of the RD problem. First, they assumed that every province of the Roman Empire is safe if there is at least one legion stationed within it and every unsafe province is defended if it is adjacent to a safe province. Then they required that for every unsafe province there exists at least one adjacent safe province whose legion could move and protect it in case it is attacked, such that this particular legion movement does not affect the Empire's safety, i.e., all provinces are considered to be defended before and after the movement.

Similarly as for the RD problem, for a graph $G = (V, E)$ and a function $f : V \to \{0, 1, 2\}$, every vertex with positive weight is considered to be defended, and a vertex u with property $f(u) = 0$ is considered to be defended if it is adjacent to a vertex $v \in V$ with positive weight. A function f is called a weak Roman dominating function (WRD function) on a graph G if every vertex u with property $f(u) = 0$ is adjacent to a vertex v with property $f(v) > 0$ and, with respect to the function f', $f' : V \to \{0, 1, 2\}$ defined by $f'(u) = 1$, $f'(v) = f(v) - 1$ and $f'(w) = f(w)$, $w \in V \setminus \{u, v\}$, all vertices are defended. The problem of finding the WRD function f with the minimal weight for a given graph G is referred to as the weak Roman domination problem (WRD problem). The minimum weight of the WRD function f, denoted by $\gamma_r(G)$, represents the weak Roman domination number.

We illustrate the weak Roman domination problem in the example below.

Example 2. Let us assume that the Roman Empire can be described by the graph $G = (V, E)$ presented on Figure 1. The optimal solution value for the WRD problem on the given graph is 3. Legions are stationed such that provinces represented by vertices v_1, v_5 and v_7 are with one stationed legion while all other provinces are without stationed legions, see Figure 3 (vertices are marked by red circles if they are representing provinces with one stationed legion and marked by white circles if they are representing provinces without stationed legions).

With the given strategy, in case of an attack, provinces represented by vertices v_2 and v_8 are defended by the legion stationed at the province represented by the vertex v_1. In case of attack, movements of legion stationed at province v_1 to province v_2 or to v_8 does not affect Empire's safety. Similarly, provinces v_4 and v_6 are defended

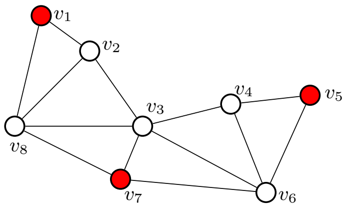


Figure 3: Illustrated solution of the WRD problem for a graph G defined in the Example 2

by the legion from province v_5, province v_3 is defended by the legion from province v_7, etc.

Ivanović [6] showed that neither the CPLEX nor the Gurobi optimization solvers were able to solve the WRD problem on a huge number of instances with more than 100 vertices. Since there is only one algorithm for solving the WRD problem (see [7]), which is written only for block graphs, we present a Variable Neighborhood Search solution for solving the WRD problem on any types of graphs.

We also show that the same algorithm can be applied to the RD problem, although Burger et al. [8] showed that there are significant differences in solving these two problems (their assumption was based on the fact that the RD problem involves static configuration of legions on the vertices of G, while the WRD problem involves moving a legion between the adjacent vertices).

This paper is organized as follows. Previous work is given in Section 2. The Variable Neighborhood Search algorithm is proposed in Section 3. Computational results are summarized in Section 4.

2 PREVIOUS WORK

The Roman domination problem was introduced by Stewart [9] and ReVelle and Rossing [1]. Inspired by Stewart's paper, Cockayne et al. [2] gave some properties of the Roman domination sets. Later Henning et al. [3] introduced the WRD problem as special variant of the RD problem and observed that every RD function in a graph G is also a WRD function in G. In the same paper they proved relation $\gamma(G) \leq \gamma_r(G) \leq \gamma_R(G) \leq 2\gamma(G)$, where $\gamma(G)$ represents cardinality of the minimum dominating set on the graph G (dominating set is a set of vertices such that each of the other vertices has a neighbor in the dominating set). Relations between several different domination numbers were summarized by Chellali et al. [10].

Upper and lower bounds for γ_R for special types of graphs were determined, for instance, in [2, 11, 13, 14, 15, 16, 17]. Exact values for γ_R for paths, cycles, complete, complete n-partite and Petersen $P(n, 2)$ graphs were given in [2, 11, 15, 16, 18, 19, 20, 21, 22], while cardinal and Cartesian products of paths and cycles and lexicographic product of some graphs were given in [15, 16, 19]. Exact values of the $\gamma_r(G)$ for paths, cycles, complete, complete n-partite, $2 \times n$ grid and web graphs and values of $\gamma_r(G)$ of corona and products of some special types of graphs were given in [3, 12, 23, 24].

The complexity of computing γ_R when restricted to interval graphs was mentioned as an open question in [2]. In the same paper it was shown that the problem of computing γ_R on trees can be solved in linear time and that it remains NP-complete even when restricted to split graphs, bipartite graphs, and planar graphs. Linear-time algorithm for computing γ_R on bounded tree-width graphs was proposed in [25]. In [20] it was shown that γ_R can be computed in linear time on interval graphs and co-graphs. In the same papers, the authors give a polynomial time algorithm for computing γ_R on AT-graphs and graphs with d-octopus. Linear-time approximation algorithm and a polynomial time approximation scheme for the RD problem on unit disk graphs was given in [22]. If we assume that the size of G is a given constant, Pavlič and Žerovnik provided algorithm for computing γ_R for polygraphs, including rota-graphs and fascia-graphs, that run in constant time in [19]. Some variants of the algorithm for solving the RD problem on a grid graph together with theoretical properties of γ_R of grid graphs were given in [13]. In [13] Currò also showed that the same algorithm can be applied to some other types of graph.

A binary programing formulations for the RD problem, which can be used for computing γ_R on arbitrary graphs by using standard optimization solvers, were provided by ReVelle and Rossing [1] and Burger et al. [4]. Burger et al. [4] also gave a binary programming formulations for the WRD problem. Recently Ivanović [6] gave another formulation for the WRD problem. Ivanović compared formulations for the WRD problem in [6], showing that neither CPLEX nor Gurobi optimization solvers were able to solve the WRD problem, regardless of the used formulation, on many instances with more than 100 vertices.

Peng [7] gave a linear time algorithm for computing γ_r on block graphs. Providing two faster algorithms, Chapelle et al. [26] broke trivial enumeration barrier of $O^*(3^n)$ for calculating $\gamma_r(G)$ (the notation $O^*(f(n))$ suppresses polynomial factors). With the first algorithm they proved that the WRD problem can be solved in $O^*(2^n)$ time needing exponential space. The second algorithm uses polynomial space and time, $O^*(2.2279^n)$.

For some special classes of graphs (interval graphs, intersection graphs, co-graphs and distance-hereditary graphs) the RD problem can be solved in linear time [15], but in the general case, the RD problem is NP-complete, [11]. Proof that the WRD problem is NP-complete, even when restricted to bipartite and chordal graphs, is given in [3].

Now, since both the Roman and the weak Roman domination problems are NP-complete problems, creating a heuristic that could be successful in finding an optimal solution value, providing legions schedule as well, represents a challenge.

Therefore, in [13] a genetic algorithm for solving the RD problem was proposed by Currò, and that was the only heuristic written for any type of Roman domination problem known to the authors. In the mentioned paper, the author proposes a set of instances on random generated graphs which will be used in experimental results of this paper.

In the next section we propose the Variable Neighborhood Search algorithm for solving both the Roman and the weak Roman domination problems on graphs. The VNS heuristic is chosen because it was previously proven to be successful for some problems on graphs, for example [27, 28].

3 VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING ROMAN AND WEAK ROMAN DOMINATION PROBLEMS

The Variable Neighborhood Search (VNS) is a heuristic method, which starts from some point from the search space, explores its neighborhoods, then changes the starting point through some search procedures such that it moves to another point of the search space, explores its neighborhoods, and repeats the whole procedure in order to find a better solution. The VNS heuristic was proposed by Mladenović [29] and later studied by Mladenović and Hansen [30] and Hansen and Mladenović in [31].

With respect to the problems' definitions, let us assume that all Roman provinces are represented by a set of vertices V, $n = |V|$, and all existing roads by the set of edges $E = \{e = (i,j), \quad i,j \in V,\ i \text{ and } j \text{ are connected}\}$, $m = |E|$, of some simple undirected graph $G = (V,E)$. Given that graph G is undirected, we will say that $e = (i,j) \in E$ implies $(j,i) \in E$. Moreover, for every vertex $i \in V$ let the set of all vertices adjacent to the vertex i be marked by N_i. Furthermore, let us assume that each province is represented by a number $i = 1,\dots,n$, and the number of legions stationed within a province i is represented by value x_i. Vector $X = (x_1,\dots,x_n)$ of values x_i, $i = 1,\dots,n$, is a feasible solution to the RD problem (WRD problem) if f, $f : V \to \{0,1,2\}$ defined by

$$f(i) = x_i, \quad i \in V \tag{1}$$

is a Roman domination function (weak Roman domination function).

Given that a feasible solution to the WRD problem does not have to be a feasible solution to the RD problem, we define a function *feasibleSolution*$(X, \textit{problem})$ which checks if X is a feasible solution for the *problem* $\in$ {RD, WRD}.

In order to check if vector X is a feasible solution to the RD problem, for every element x_i $(i = 1,\dots,n)$ *feasibleSolution(X, RD)* checks if x_i is a positive value, or $x_i = 0$ and there is at least one vertex v_j connected to v_i such that $x_j = 2$.

In order to check if vector X is a feasible solution to the WRD problem, for every element x_i $(i = 1,\dots,n)$ *feasibleSolution(X, WRD)* checks if it is a positive value, or $x_i = 0$ and at least one of the following two conditions holds:

1. there exists at least one element x_j $(j = 1,\dots,n,\ j \neq i)$ with properties $x_j = 2$ and $j \in N_i$, i.e.

- after a single legion movement from a province j to a province s $(s \neq i, j)$ there still is one legion stationed at a province j which defends provinces i and j;
- after a single legion movement from a province j to a province i, both provinces i and j are defended by stationed legions.

2. there exists at least one element x_j, $j \in N_i$, such that $x_j = 1$ and swapping the values of x_i and x_j does not affect the feasibility of the vector X. More precisely, after the swap, for every element x_s, $s \in N_j$, with property $x_s = 0$, there exists at least one x_k, $k \in N_s, k \neq j$, with property $x_k > 0$, i.e.

   - in order to move a single legion from a province j to a province i, all provinces s, which are neighbors with j and which are without any stationed legion, must have another neighbor k $(k \neq j)$ with at least one stationed legion.

We will say that the function *feasibleSolution(X, problem)* is satisfied if there are no undefended provinces with respect to the *problem*.

Also, we create function *penalty(X, problem)*, which calculates the number of undefended provinces with respect to the *problem*.

Further, we will say that two solutions, X and X', have difference of the first order if one legion was moved from one province to another (value of one element, with value lower than 2, of the vector X, is increased by one, while value of the other element, with positive value, of the vector X, is decreased by one) or disbanded (value of one element, with positive value, of the vector X, is decreased by one). Respectively, two solutions have difference of the k^{th} order if at most k legions were moved, including possible disbanding.

Now, let us define a set $\mathcal{N}_k(X)$, $k = k_{min}, \ldots, k_{max}$ as the set of all vectors X' that have difference of the k^{th} order from the solution X and call that set k^{th} *Neighborhood to the solution X*.

The VNS-based heuristic can be defined in such a way that it starts from the *initial* feasible solution X, *shakes* it by creating another solution $X' \in \mathcal{N}_k(X)$ (by the expression *shake* we mean *movement of a certain number of legions*) and then applies *local search method* in order to create a better feasible solution X''. If the feasible solution X'', obtained by the local search procedure, is not better than the current incumbent X $(F(X'') \geq F^*)$, the VNS algorithm repeats the procedure of shaking, but in neighborhood $\mathcal{N}_{k+k_{step}}(X)$ (i.e., k increments by k_{step}) and local search within it and so on until k reaches its maximum k_{max}. Otherwise, if $F(X'') < F^*$, X^* becomes X'', F^* becomes $F(X'')$ and k becomes k_{min}. Changing neighborhoods enables one to get out from the local minima. The VNS algorithm is presented as Algorithm 1. Functions *InitialSolution()*, *Shake()*, *LocalSearch()* and *StoppingCondition()* are described below.

Function *InitialSolution*() (pseudo code is presented as Algorithm 2) is defined so that it produces an initial feasible solution X^* by applying random changes to elements of the zero vector X. That is, *InitialSolution*() assigns randomly generated number from the set $\{1, 2\}$ to a randomly chosen element of the vector X until X

Algorithm 1 Variable Neighborhood Search metaheuristic

```
1: $X^* \leftarrow \mathit{InitialSolution}()$;
2: $F^* \leftarrow F(X^*)$;
3: repeat
4:     $k \leftarrow k_{min}$;
5:     repeat
6:         $X \leftarrow X^*$;
7:         $X' \leftarrow \mathit{Shake}(X, k)$;
8:         $X'' \leftarrow \mathit{LocalSearch}(X')$;
9:         if $F(X'') < F^*$ then
10:            $F^* \leftarrow F(X'')$;
11:            $X^* \leftarrow X''$;
12:            $k \leftarrow k_{min}$;
13:        else
14:            $k \leftarrow k + k_{step}$;
15:    until $k > k_{max}$
16: until $\mathit{StoppingCondition}()$
```

Algorithm 2 *InitialSolution*()

```
1: $X \leftarrow \{0, \ldots, 0\}$;
2: repeat
3:     $i \leftarrow$ random number $\in \{1, \ldots, n\}$;
4:     $x_i \leftarrow$ random number $\in \{1, 2\}$;
5: until $(\mathit{feasibleSolution}(X, \mathit{problem}))$
6: for $i = 1, \ldots, n$ do
7:     if $x_i > 0$ then
8:         $x_i \leftarrow x_i - 1$;
9:         if not$(\mathit{feasibleSolution}(X, \mathit{problem}))$ then
10:            $x_i \leftarrow x_i + 1$;
```

becomes a feasible solution. Then, given that the function *InitialSolution*() finds a feasible solution, and our goal is to find a feasible solution such that the objective function value $F(X)$ ($F(X) = \sum_{i=1}^{n} x_i$) is minimal, the found solution will be, for now, saved as the best one ($X^* \leftarrow X$, $F^* \leftarrow F(X^*)$).

Further, in order to lower the value F^*, i.e., to improve the incumbent, among the elements of the vector X with positive value, *InitialSolution*() searches for an element whose value could be decreased by one such that the resulting vector remains a feasible solution. If such an element is found, *InitialSolution*() will decrease its value by one, and then continue to search for an element of the incumbent with the same property. Whenever the procedure of decreasing a value of one element produces a feasible vector, the resulting vector will be stored as the best one and objective function value $F(X)$ will be stored as F^*. This procedure repeats until there are no elements whose decreased value will result with feasible X.

Algorithm 3 *Shake*()

```
1: $X \leftarrow X^*$
2: $DecreasingProcedure(X)$;
3: for $j = 1, \dots, k$ do
4:     $a \leftarrow$ random number $\in \{1, \dots, n\}$ such that $x_a \neq 0$;
5:     $b \leftarrow$ random number $\in \{1, \dots, n\}$ such that $x_b \neq 2$;
6:     $x_a \leftarrow x_a - 1$;
7:     $x_b \leftarrow x_b + 1$;
8: if $feasibleSolution(X, problem)$ then
9:     $X^* \leftarrow X$;
10:    $DecreasingProcedure(X)$;
```

Now, if it is possible to find a feasible solution with the same or smaller objective function value than F^*, the resulting solution will be better than the current incumbent. Hence, we define the following two functions, *Shake*() and *LocalSearch*(). These two functions are defined to search for a better feasible solution than the one with which they start the searching process.

Therefore, $Shake(X^*, k)$ function (presented as Algorithm 3) starts with a feasible solution X^*, stores it as X ($X \leftarrow X^*$) and then randomly chooses an element of the solution X with positive value and decreases its value by one. If the resulting vector is again a feasible solution, it stores it as the new best solution and repeats the process until an infeasible solution is found. We call this process *DecreasingProcedure*(). Then, among the elements of the current solution X with value lower than 2, shake function randomly choses one element, and among the elements with positive value of the incumbent X, it randomly chooses another element and increases a value of the first chosen element by one and decreases the value of the second chosen element also by one (i.e., it moves one legion) and repeats this process k times. If the resulting vector X' is a feasible one, given that $F(X') < F^*$ the new best feasible is found. Therefore, X' will be stored as the new best feasible ($X^* \leftarrow X'$). Also, if $X^{'}$ is feasible, we will apply *DecreasingProcedure*() to the vector $X^{'}$ and resulting vector denote as X' (note that in this case it follows that $F(X') \leq F^* - 1$).

Now, the *LocalSearch*(X') function (presented as Algorithms 4 and 5) starts with an infeasible incumbent X', calculates its *penalty*$(X', problem)$ value and stores it as nd_{min}. Then it searches a neighborhood $\mathcal{N}_1(X')$ of the incumbent X' in order to find a feasible solution. If a solution with lower penalty value is found it will be stored as incumbent and search for a better solution continues. If a solution with penalty value equal to zero is found, it means that a feasible solution is found. If there is no solution with penalty value lower or equal to nd_{min} within the neighborhood $\mathcal{N}_1(X')$ of the incumbent, local search procedure will continue its search in the neighborhood $\mathcal{N}_2(X')$ of the incumbent. In both cases, whenever a feasible solution is found, it will be stored as the new best feasible solution. Also, local search procedure will continue to search for a feasible solution within the neighborhoods of the incumbent

Algorithm 4 *LocalSearch*()

```
1: $nd_{min} \leftarrow penalty(X', problem)$;
2: while some improvement is made do
3:     for $i = 1, \dots, n$ such that $x_i^{'} > 0$ do
4:         $x_i^{'} \leftarrow x_i^{'} - 1$;
5:         if $feasibleSolution(X', problem)$ then
6:             $X^* \leftarrow X'$;
7:             $DecreasingProcedure(X')$;
8:             $nd_{min} \leftarrow penalty(X', problem)$;
9:             go to line 3;
10:        else
11:            for $j = 1, \dots, n,\ j \neq i$ such that $x_j^{'} < 2$ do
12:                $x_j^{'} \leftarrow x_j^{'} + 1$;
13:                $nd \leftarrow penalty(X', problem)$;
14:                if $nd = 0$ then
15:                    execute lines 6-9;
16:                else
17:                    if $nd < nd_{min}$ then
18:                        $X_{better}^{'} \leftarrow X'$;
19:                        $nd_{min} \leftarrow nd$;
20:                    if $nd = nd_{min}$ then
21:                        $X_{same}^{'} \leftarrow X'$ with some probability;
22:                $x_j^{'} \leftarrow x_j^{'} - 1$;
23:        $x_i^{'} \leftarrow x_i^{'} + 1$;
24:    if $X_{better}^{'}$ is found then
25:        $X^{'} \leftarrow X_{better}^{'}$;
26:    else
27:        if $X_{same}^{'}$ is found then
28:            with some probability $X^{'} \leftarrow X_{same}^{'}$;
29:        else
30:            run $LS2()$;
31: $X'' \leftarrow X^*$;
```

(i.e., a decreasing procedure will be applied to the feasible incumbent) until there is no better feasible solution.

In other words, local search procedure consists of three steps. In the first step, local search procedure searches for an element (of the incumbent X') with positive value, decreases its value by one and checks if the resulting vector is a feasible one. If the resulting vector is a feasible solution, it will be stored as X^*. If the resulting vector is infeasible, the procedure goes to the second step of the local search. In the second step, the local search procedure searches for an element $x_j^{'}$ of the incumbent of the local search procedure with property $x_j^{'} < 2$, such that increasing its value by one creates a feasible solution. If the required element is found, its value will be increased by one and the resulting feasible solution stored as X^*. If a feasible solution is found (both in the first and in the second step), $DecreasingProcedure()$ will be applied to that feasible incumbent, nd_{min} will be set to be equal to $penalty(X', problem)$ and the local search procedure will restart from the beginning of the first step (lines 6-9 and 15 of Algorithm 4). If the required element of the second step was not found, solution with the smallest penalty value $penalty(X', problem)$ will be stored as $X^{'}_{better}$ and the solution with the penalty value equal to the incumbent will be stored as $X^{'}_{same}$. Then, when the second step is finished, in case that a better solution than the incumbent is found, it will be set as the incumbent solution and the second step will restart from the beginning. Similarly, if at least a solution of the same quality is found, it will be set as the incumbent solution with some probability and the second step will restart from the beginning. Otherwise, if there is no better solution nor a solution of the same quality, the third step of the local search procedure will start.

In the third step of the local search procedure, we explore a neighborhood $\mathcal{N}_2(X')$ of the incumbent in order to find a feasible solution. We denoted the third step of the local search procedure as $LS2()$ only because we want to make algorithm of $LocalSearch()$ function easier for reading.

In the third step (which is presented as Algorithm 5), the local search procedure searches for an element $x^{'}_i$ with value $x^{'}_i = 2$ and for an element $x^{'}_j$ with value $x^{'}_j < 2$ $(i, j = 1, \dots, n)$. Then, it decreases the value of $x^{'}_i$ by two and increases a value of $x^{'}_j$ by one and then checks if a feasible solution is found, or if there exists an element $x^{'}_s < 2$ such that increasing its value by one results with a feasible solution or with a better infeasible solution. Similarly as in the first two steps, $LS2()$ function computes *penalty()* value before and after each change and stores an incumbent solution X' with smaller penalty value than nd_{min} as $X^{'}_{better}$ and the incumbent with the same penalty value as $X^{'}_{same}$. Again, whenever a better incumbent is found, nd_{min} will be set to be equal to *penalty($X^{'}_{better}$, problem)* and the incumbent solution of the same quality will be stored with some probability. Then, if a process of decreasing a value of an element $x^{'}_i$ by two and increasing a value of each pair of elements $x^{'}_j$ and $x^{'}_s$ by one does not create a feasible solution, values of elements x_i, x_j and x_s will be restored and the third step will continue its search with the next element whose value is equal to 2. In case that all element combinations are checked and better solution is found, it will be set as the incumbent and $LS2()$ will restart its search within the new incumbent. Similarly, in case that all elements combinations are checked and only a solution of the same quality is found, it will be set as the incumbent with some probability and $LS2()$ will restart.

During all the steps of the local search procedure we are also checking if moves from one solution to the solution of the same quality will not make a loop, i.e., we will not store the incumbent of the same quality if it will take us to some previous incumbent. Given that the size of a loop may vary, we do not allow moves from one incumbent to the incumbent of the same quality for more then k_{max} successive times. This means that the second and the third step will restart with the solution

Algorithm 5 $LS2()$

```
1: $nd_{min} \leftarrow penalty(X', problem)$
2: while some improvement is made do
3:     for $i = 1, \dots, n$ such that $x_i^{'} = 2$ do
4:         $x_i^{'} \leftarrow x_i^{'} - 2$
5:         for $j = 1, \dots, n$ such that $x_j^{'} < 2$ do
6:             $x_j^{'} \leftarrow x_j^{'} + 1$
7:             if $feasibleSolution(X', problem)$ then
8:                 $X^* \leftarrow X'$
9:                 $DecreasingProcedure(X')$
10:                $nd_{min} \leftarrow penalty(X', problem)$
11:                go to line 2
12:            else
13:                for $s = 1, \dots, n$, such that $x_s^{'} < 2$ do
14:                    $x_s^{'} \leftarrow x_s^{'} + 1$
15:                    if $feasibleSolution(X', problem)$ then
16:                        apply lines $8 - 11$
17:                    else
18:                        $nd \leftarrow penalty(X')$
19:                        if $nd < nd_{min}$ then
20:                            $X'_{better} \leftarrow X'$
21:                            $nd_{min} \leftarrow nd$
22:                        if $nd = nd_{min}$ then
23:                            $X'_{same} \leftarrow X'$ with some probability
24:                    $x_s^{'} \leftarrow x_s^{'} - 1$
25:            $x_j^{'} \leftarrow x_j^{'} - 1$
26:        $x_i^{'} \leftarrow x_i^{'} + 2$
27:    if $X_{better}^{'}$ is found then
28:        $X^{'} \leftarrow X_{better}^{'}$
29:    else
30:        if $X_{same}^{'}$ is found then
31:            with some probability $X^{'} \leftarrow X_{better}^{'}$
32:        else
33:            finish $LS2()$
```

of the same quality for no more than k_{max} successive times. If some improvements are made within $LS2()$, the local search procedure restarts from the beginning of the first step with the new incumbent. Finally, when all three steps are finished and no improvement is made, *LocalSearch()* function will finish its search and the feasible solution X^* will be returned as X''. Now, if a better feasible solution is obtained ($F(X^{''}) < F^*$), its objective function value will be stored ($F^* \leftarrow F(X^{''})$) and k will be set to k_{min}, otherwise k will be increased by k_{step}. The VNS algorithm continues until k reaches its maximum or some other stopping condition occurs.

Input parameters for the VNS heuristic are the *problem*, the minimal (k_{min}) and the maximal (k_{max}) numbers of neighborhoods that should be searched, the increment of the parameter k (k_{step}) and the maximum CPU time allowed (t_{max}). In our implementation *StoppingCondition()* finishes the VNS algorithm if either k_{max} or maximal CPU time allowed is reached.

The parameters used for the proposed VNS algorithm are $k_{min} = 1$, $k_{max} = 30$, $k_{step} = 1$ and $t_{max} = 7\,200$ s and probability is set to $p = 0.5$.

The VNS algorithm cannot guarantee finding global optima because of its non-deterministic nature. Therefore, in order to find solution of sufficiently high quality it is necessary to run the VNS heuristic algorithm on the same instance more than once. Hence, in out experiments each instance was run 20 times.

4 COMPUTATIONAL RESULTS

Experimental results obtained by the proposed VNS algorithm for solving the RD and the WRD problems are presented in this section. The VNS algorithm was implemented in C++. All computational experiments have been performed on Intel® Core^{TM} i7-4700MQ CPU@2.40 GHz with 8 GB RAM, under Windows 10 operating system.

CPLEX optimizations solver was run on all five formulations of the RD problem presented in [5] on grid, planar, net and randomly generated sets of graphs. The set of randomly generated graphs is the same as the one generated and proposed by Currò in [13] (names of instances consist of the number of vertices and of the probability that edge is incident to vertices expressed in percentage) while grid, net and planar sets are well known sets of graphs and also provided by Currò. Since there are several different ILP formulations of the Roman and the weak Roman domination problems (see [5] and [6]), and that performance of CPLEX differs in accordance with used ILP formulation, for the RD problem we present only instances for which optimal solution value is found, while for the WRD problem the results are presented on all instances with some known solution. In case that CPLEX was successful in finding an optimal solution value by using more than one formulation, the smallest running time is presented.

The results are summarized in Tables 1–8.

Tables 1–4 contain instances where CPLEX optimization solver was able to find and prove optimality of the found solution value for the RD problem (CPLEX was run for all five formulations of the RD problem presented in [5]). Tables 5–8 contains instances where CPLEX and Gurobi optimization solvers were successful in finding some solution value by using at least one ILP formulation presented in [6] within the given time. In all tables, whenever the optimal solution value is found by more than one formulation, the smallest running time is shown. Also, whenever optimization solver was unable to prove optimality of the found solution either because of time limit or "out of memory" status, in the column t_{sol} we put sign "–".

Instances are sorted by the number of vertices and the number of edges, in that order. Tables are organized as follows: The name of the instance is given in the first column. The next two columns ($|V|$, $|E|$) represent the number of vertices and the number of edges. In tables that correspond to the RD problem for all instances we have optimal solution values. Therefore, in the next two columns, *opt* and t_{cpl}, optimal solution value and minimal running time are given. In tables that correspond to the WRD problem we have three columns, the optimal solution value, the best solution value and the smallest running time, which is given regardless the optimization solver and ILP formulation. It should be noted that for the WRD problem optimal solution values and minimal running times of standard optimization solvers are taken from [6]. Also note that, in case that optimization solver could not provide an optimal solution value, a symbol "-" stands in the column t_{sol}.

For both problems, the VNS algorithm was run 20 times for each problem instance and informations of the best solution values obtained in these 20 runs are given in the final four columns (sol, t, err, σ) of all the tables. The best solution value obtained by the VNS algorithm is given in the column *sol* and whenever the VNS solution value was equal to the optimal solution value (from *opt* column), it was marked as "*opt*". The best time in 20 runs, necessary for the VNS algorithm to reach the corresponding solution in the first occurrence is given in the column t. The final two columns err and σ contains informations on the average solution quality: err stands for average relative error of found solutions from the best found solution, which is calculated as $err = \frac{1}{20}\sum_{i=1}^{20} err_i$, where $err_i = |\mathrm{VNS}_i - sol|/|\mathrm{VNS}_i|$, and VNS_i is the VNS solution obtained in the i^{th} run. Parameter σ is the standard deviation of the err obtained by the formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(err_i - err)^2}$.

The VNS algorithm for the RD problem is tested on 231 different instances and achieves the optimal solution on 218 of them. All solutions are found within the time limit (running time for 99 instances is lower than 1 second and only for 29 larger than 100 seconds). For majority of instances (on 214 instances), percentage average relative error from the found solution is lower than 2.5 %. Also, for the majority of instances (for 121 instances) the VNS heuristic running time is lower than the best CPLEX running time. Detailed informations of these testings are given in Tables 1–4.

Instance			CPLEX		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*opt*	t_{cpl}	*sol*	t	err	σ
grid04x10	40	66	20	0.081	opt	0.01	0	0
grid05x08	40	67	21	0.081	opt	0.005	0	0
grid08x05	40	67	21	0.062	opt	< 0.01	0	0
grid10x04	40	66	20	0.077	opt	0.013	0	0
grid03x14	42	67	22	0.042	opt	< 0.01	0	0
grid06x07	42	71	22	0.119	opt	< 0.01	0	0
grid07x06	42	71	22	0.115	opt	< 0.01	0	0
grid14x03	42	67	22	0.062	opt	0.031	0	0

Table 1 continues ...

Instance			CPLEX		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid04x11	44	73	22	0.057	opt	0.013	0	0
grid11x04	44	73	22	0.046	opt	< 0.01	0	0
grid03x15	45	72	24	0.046	opt	0.012	0	0
grid05x09	45	76	23	0.168	opt	< 0.01	0	0
grid09x05	45	76	23	0.148	opt	0.013	0	0
grid15x03	45	72	24	0.061	opt	0.022	0	0
grid04x12	48	80	24	0.058	opt	0.034	0	0
grid06x08	48	82	24	0.05	opt	0.033	0.0040	0.0120
grid08x06	48	82	24	0.098	opt	0.012	0.0040	0.0120
grid12x04	48	80	24	0.076	opt	0.019	0	0
grid07x07	49	84	24	0.098	opt	0.029	0.0060	0.0143
grid05x10	50	85	26	0.147	opt	< 0.01	0	0
grid10x05	50	85	26	0.166	opt	< 0.01	0	0
grid04x13	52	87	26	0.162	opt	0.104	0	0
grid13x04	52	87	26	0.099	opt	0.017	0.0019	0.0081
grid06x09	54	93	27	0.111	opt	0.436	0.0143	0.0175
grid09x06	54	93	27	0.179	opt	< 0.01	0.0071	0.0143
grid05x11	55	94	28	0.153	opt	0.013	0	0
grid11x05	55	94	28	0.184	opt	< 0.01	0	0
grid04x14	56	94	28	0.059	opt	0.035	0.0017	0.0075
grid07x08	56	97	28	0.131	opt	< 0.01	0	0
grid08x07	56	97	28	0.153	opt	0.033	0	0
grid14x04	56	94	28	0.06	opt	0.438	0.0356	0.0109
grid04x15	60	101	30	0.092	opt	< 0.01	0.0016	0.0070
grid05x12	60	103	30	0.13	opt	0.036	0.0194	0.0158
grid06x10	60	104	30	0.092	opt	0.041	0.0048	0.0115
grid10x06	60	104	30	0.152	opt	0.163	0.0097	0.0148
grid12x05	60	103	30	0.177	opt	0.078	0.0129	0.0158
grid15x04	60	101	30	0.075	opt	0.04	0	0
grid07x09	63	110	31	0.066	opt	0.135	0.0094	0.0143
grid09x07	63	110	31	0.162	opt	0.082	0	0
grid08x08	64	112	32	0.118	opt	0.031	0.0015	0.0066
grid05x13	65	112	33	0.173	opt	0.171	0.0029	0.0088
grid13x05	65	112	33	0.204	opt	0.054	0.0044	0.0105
grid06x11	66	115	33	0.137	opt	0.045	0.0059	0.0118
grid11x06	66	115	33	0.169	opt	0.246	0.0029	0.0088
grid05x14	70	121	35	0.207	opt	0.264	0.0083	0.0127
grid07x10	70	123	34	0.146	opt	0.164	0.0171	0.0140
grid10x07	70	123	34	0.119	opt	0.699	0.0171	0.0165
grid14x05	70	121	35	0.191	opt	0.19	0.0083	0.0127
grid06x12	72	126	36	0.169	opt	0.198	0.0054	0.0108
grid08x09	72	127	35	0.153	opt	0.017	0.0069	0.0120
grid09x08	72	127	35	0.125	opt	0.037	0.0110	0.0181

Table 1 continues . . .

Instance			CPLEX		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*opt*	t_{cpl}	*sol*	*t*	*err*	σ
grid12x06	72	126	36	0.186	opt	0.161	0.0014	0.0059
grid05x15	75	130	38	0.214	opt	0.352	0.0026	0.0077
grid15x05	75	130	38	0.247	opt	0.101	0	0
grid07x11	77	136	38	0.169	opt	0.094	0.0013	0.0056
grid11x07	77	136	38	0.186	opt	0.102	0.0026	0.0077
grid06x13	78	137	38	0.148	opt	1.553	0.0242	0.0148
grid13x06	78	137	38	0.209	opt	38	0.0256	0.0079
grid08x10	80	142	39	0.128	opt	0.132	0.0113	0.0124
grid10x08	80	142	39	0.142	opt	0.039	0.0075	0.0115
grid09x09	81	144	38	0.073	opt	2.937	0.0226	0.0236
grid06x14	84	148	41	0.134	opt	22.542	0.0306	0.0128
grid07x12	84	149	41	0.168	opt	1.727	0.0083	0.0114
grid12x07	84	149	41	0.192	opt	1.062	0.0024	0.0071
grid14x06	84	148	41	0.231	opt	7.766	0.0225	0.0116
grid08x11	88	157	42	0.192	opt	11.391	0.0275	0.0151
grid11x08	88	157	42	0.141	opt	0.778	0.0206	0.0187
grid06x15	90	159	44	0.247	opt	5.733	0.0188	0.0125
grid09x10	90	161	43	0.223	opt	1.224	0.0279	0.0184
grid10x09	90	161	43	0.237	opt	0.672	0.0102	0.0133
grid15x06	90	159	44	0.264	opt	3.141	0.0133	0.0109
grid07x13	91	162	44	0.178	opt	0.801	0.0177	0.0112
grid13x07	91	162	44	0.178	opt	0.882	0.0177	0.0131
grid08x12	96	172	46	0.21	opt	1.527	0.0178	0.0176
grid12x08	96	172	46	0.191	opt	5.175	0.0159	0.0113
grid07x14	98	175	47	0.247	opt	1.621	0.0247	0.0149
grid14x07	98	175	47	0.214	opt	2.929	0.0196	0.0137
grid09x11	99	178	47	0.194	opt	3.737	0.0124	0.0136
grid11x09	99	178	47	0.287	opt	4.522	0.0245	0.0187
grid10x10	100	180	48	0.22	opt	0.199	0.0051	0.0088
grid08x13	104	187	50	0.262	opt	0.274	0.0097	0.0130
grid13x08	104	187	50	0.401	opt	9.993	0.0146	0.0135
grid07x15	105	188	50	0.348	opt	20.739	0.0252	0.0121
grid15x07	105	188	50	0.278	opt	22.274	0.0243	0.0102
grid09x12	108	195	51	0.268	opt	9.665	0.0244	0.0204
grid12x09	108	195	51	0.29	opt	22.53	0.0208	0.0183
grid10x11	110	199	52	0.306	opt	2.545	0.0185	0.0192
grid11x10	110	199	52	0.256	opt	12.061	0.0222	0.0188
grid08x14	112	202	53	0.289	opt	6.864	0.0201	0.0138
grid14x08	112	202	53	0.284	opt	1.213	0.0228	0.0159
grid09x13	117	212	55	0.232	opt	10.045	0.0260	0.0224
grid13x09	117	212	55	0.439	opt	46.869	0.0262	0.0184
grid08x15	120	217	57	0.404	opt	5.05	0.0154	0.0119
grid10x12	120	218	56	0.236	opt	29.077	0.0381	0.0228

Table 1 continues . . .

Instance			CPLEX		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid12x10	120	218	56	0.326	opt	27.666	0.0343	0.0150
grid15x08	120	217	57	0.414	opt	18.631	0.0161	0.0155
grid11x11	121	220	57	0.443	opt	2.414	0.0219	0.0198
grid09x14	126	229	58	0.25	opt	0.518	0.0397	0.0277
grid14x09	126	229	58	0.334	opt	46.688	0.0458	0.0170
grid10x13	130	237	61	0.519	opt	12.797	0.0174	0.0178
grid13x10	130	237	61	0.529	opt	1.85	0.0188	0.0226
grid11x12	132	241	62	0.453	opt	26.008	0.0248	0.0183
grid12x11	132	241	62	0.464	opt	31.964	0.0204	0.0131
grid09x15	135	246	63	0.535	opt	36.088	0.0252	0.0185
grid15x09	135	246	63	0.733	opt	23.271	0.0312	0.0168
grid10x14	140	256	65	0.478	opt	78.302	0.0339	0.0165
grid14x10	140	256	65	0.432	opt	10.337	0.0359	0.0210
grid11x13	143	262	66	0.463	opt	70.571	0.0303	0.0223
grid13x11	143	262	66	0.503	opt	21.158	0.0372	0.0250
grid12x12	144	264	67	0.516	opt	36.922	0.0349	0.0185
grid10x15	150	275	70	0.715	opt	126.053	0.0266	0.0221
grid15x10	150	275	70	0.951	opt	24.143	0.0301	0.0189
grid11x14	154	283	71	0.483	opt	59.802	0.0438	0.0246
grid14x11	154	283	71	0.67	opt	62.236	0.0382	0.0203
grid12x13	156	287	72	0.715	**73**	115.106	0.0232	0.0159
grid13x12	156	287	72	0.783	opt	62.928	0.0384	0.0168
grid11x15	165	304	76	0.77	opt	117.803	0.0484	0.0198
grid15x11	165	304	76	0.918	opt	52.315	0.0406	0.0193
grid12x14	168	310	77	0.614	opt	181.88	0.0389	0.0205
grid14x12	168	310	77	0.721	opt	155.635	0.0424	0.0222
grid13x13	169	312	78	0.77	opt	68.571	0.0325	0.0191
grid12x15	180	333	82	0.94	**83**	164.384	0.0362	0.0191
grid15x12	180	333	82	1.3	**83**	130.71	0.0439	0.0207
grid13x14	182	337	83	0.777	opt	75.472	0.0486	0.0249
grid14x13	182	337	83	0.776	opt	201.98	0.0441	0.0270
grid13x15	195	362	89	1.73	opt	407.358	0.0483	0.0277
grid15x13	195	362	89	1.309	opt	139.451	0.0460	0.0239
grid14x14	196	364	88	0.739	opt	353.878	0.0516	0.0254
grid14x15	210	391	95	1.198	opt	282.147	0.0508	0.0250
grid15x14	210	391	95	1.159	opt	92.424	0.0543	0.0202
grid15x15	225	420	102	1.357	opt	697.859	0.0536	0.0240
grid20x20	400	760	176	37.579	**185**	676.713	0.0390	0.0135
grid30x20	600	1 150	260	1 279.438	**286**	5 114.624	0.0330	0.0160

Table 1. Experimental results for the RD problem on grid graph instances

From Table 1 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful on 5 among 133 instances of grid type). On instances “grid12x13”, “grid12x15”, “grid15x12”, “grid20x20” and “grid30x20”, where an optimal solution was not reached, percentage average relative error from the found solution is lower than 2.1 %. Further, on 123 of 133 instances, percentage average relative error from the found solution is lower or equal to 2.5 % and on 5 instances between 2.5 % and 3 %. So, from Table 1 we can conclude that for the RD problem on grid graph instances the VNS algorithm provides solutions of good quality and within the time limit.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	*err*	σ
plan10	10	27	3	0.048	opt	< 0.01	0	0
plan20	20	105	5	0.062	opt	< 0.01	0	0
plan30	30	182	5	0.046	opt	< 0.01	0	0
plan50	50	465	6	0.082	opt	< 0.01	0	0
plan100	100	1 540	10	0.0383	opt	0.054	0	0
plan150	150	2 867	12	1.303	opt	1.166	0	0
plan200	200	4 475	16	145.262	opt	2.466	0	0

Table 2: Experimental results for the RD problem on planar graph instances

From Table 2 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on all instances with σ equal to zero. The VNS algorithm was also tested on instances “plan250” and “plan300” but, because CPLEX was unable to provide optimal solution values on these instances, we will not present the VNS algorithm results for these instances either. Also, we can conclude that instances of planar type are easier for solving for the VNS algorithm than for CPLEX, given the fact that the VNS algorithm provides results much more rapidly.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	*err*	σ
Net-10-10	100	342	28	0.043	opt	0.129	0	0
Net-10-20	200	712	56	0.088	opt	18.013	0.0018	0.0053
Net-20-20	400	1 482	98	0.134	opt	944.94	0.0228	0.0316
Net-30-20	600	2 252	140	0.162	**145**	6916.4	0.0580	0.0274

Table 3: Experimental results for the RD problem on net graph instances

From Table 3 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on 3 of 4 instances. On instance “Net-30-20”, where an optimal solution value was not reached, percentage average relative error is equal to 2.74 %. Instances of the net type can be considered as easy for solving for CPLEX given the fact that CPLEX is able to provide results for less than 1 second.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	*err*	σ
Random-50-1	50	49	32	0.062	opt	0.031	0	0
Random-50-2	50	49	33	0.062	opt	0.069	0	0
Random-50-3	50	58	28	0.084	opt	0.029	0	0
Random-50-4	50	54	30	0.08	opt	0.006	0	0
Random-50-5	50	67	28	0.1	opt	0.005	0	0
Random-50-6	50	86	25	0.184	opt	0.041	0	0
Random-50-7	50	84	26	0.1	opt	< 0.01	0	0
Random-50-8	50	95	23	0.121	opt	< 0.01	0	0
Random-50-9	50	108	23	0.152	opt	0.011	0	0
Random-50-10	50	112	22	0.162	opt	0.021	0	0
Random-50-20	50	248	12	0.337	opt	< 0.01	0	0
Random-50-30	50	373	9	0.178	opt	< 0.01	0	0
Random-50-40	50	475	8	0.432	opt	< 0.01	0	0
Random-50-50	50	597	6	0.285	opt	< 0.01	0	0
Random-50-60	50	739	4	0.115	opt	< 0.01	0	0
Random-50-70	50	860	4	0.121	opt	< 0.01	0	0
Random-50-80	50	980	4	0.131	opt	< 0.01	0	0
Random-50-90	50	1 103	3	0.131	opt	< 0.01	0	0
Random-100-1	100	100	61	0.062	opt	4.662	0.0056	0.0092
Random-100-2	100	109	59	0.1	opt	2.744	0.0058	0.0095
Random-100-3	100	181	48	0.168	opt	3.767	0.0142	0.0113
Random-100-4	100	206	45	0.438	opt	0.895	0.0184	0.0103
Random-100-5	100	231	39	0.469	opt	3.425	0.0243	0.0251
Random-100-6	100	321	34	0.532	opt	3.572	0.0157	0.0142
Random-100-7	100	317	32	0.585	opt	3.291	0.0152	0.0152
Random-100-8	100	398	29	0.774	opt	0.669	0.0017	0.0073
Random-100-9	100	430	27	0.728	opt	0.389	0	0
Random-100-10	100	498	24	1.263	opt	3.95	0.0160	0.0196
Random-100-20	100	981	14	0.971	opt	0.086	0	0
Random-100-30	100	1 477	11	2.916	opt	0.137	0.0083	0.0250
Random-100-40	100	1 945	8	0.761	opt	0.052	0	0
Random-100-50	100	2 483	7	0.808	opt	0.049	0.0188	0.0446
Random-100-60	100	2 985	6	0.345	opt	< 0.01	0	0
Random-100-70	100	3 435	5	0.285	opt	0.044	0	0
Random-100-80	100	3 935	4	0.238	opt	< 0.01	0	0
Random-100-90	100	4 446	4	0.263	opt	< 0.01	0	0
Random-150-1	150	157	94	0.115	opt	22.389	0.0011	0.0032
Random-150-2	150	243	78	0.332	opt	234.872	0.0290	0.0151
Random-150-3	150	322	65	0.834	opt	67.784	0.0171	0.0162
Random-150-4	150	437	53	1.046	opt	30.304	0.0264	0.0155
Random-150-5	150	557	46	3.115	opt	2.293	0.0169	0.0142
Random-150-6	150	705	38	10.362	opt	19.279	0.0165	0.0165

Table 4 continues ...

Instance			CPLEX		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	opt	t_{cpl}	sol	t	*err*	σ
Random-150-7	150	778	34	5.622	opt	0.462	0.0057	0.0114
Random-150-8	150	906	31	18.691	opt	0.865	0	0
Random-150-9	150	965	30	10.489	opt	3.727	0.0064	0.0161
Random-150-10	150	1 152	27	45.44	opt	3.128	0.0054	0.0128
Random-150-20	150	2 228	16	31.857	opt	1.561	0	0
Random-150-30	150	3 318	12	21.507	opt	0.383	0	0
Random-150-40	150	4 476	9	13.628	opt	0.409	0.0700	0.0458
Random-150-50	150	5 550	8	17.671	opt	0.014	0	0
Random-150-60	150	6 734	6	1.742	opt	0.012	0	0
Random-150-70	150	7 807	6	8.667	opt	0.015	0	0
Random-150-80	150	8 924	4	0.366	opt	0.019	0	0
Random-150-90	150	10 043	4	0.839	opt	0.017	0	0
Random-200-1	200	229	116	0.132	**117**	173.552	0.0167	0.0119
Random-200-2	200	390	92	0.933	**93**	647.247	0.0294	0.0184
Random-200-3	200	581	69	2.69	opt	507.393	0.0403	0.0256
Random-200-4	200	737	60	13.301	opt	568.08	0.0433	0.0214
Random-200-5	200	1 010	47	60.589	opt	41.339	0.0354	0.0217
Random-200-6	200	1 180	42	245.778	opt	84.363	0.0518	0.0332
Random-200-7	200	1 453	36	130.93	opt	11.272	0.0093	0.0173
Random-200-30	200	5 876	12	153.586	opt	9.478	0.0110	0.0346
Random-200-40	200	7 907	10	89.663	opt	0.302	0	0
Random-200-50	200	9 895	8	30.844	opt	0.248	0	0
Random-200-60	200	11 971	6	7.707	opt	0.496	0	0
Random-200-70	200	14 059	6	19.27	opt	0.025	0	0
Random-200-80	200	15 918	4	0.831	opt	0.038	0	0
Random-200-90	200	17 821	4	0.801	opt	0.03	0	0
Random-250-1	250	345	136	0.21	**137**	1 111.594	0.0220	0.0130
Random-250-2	250	633	97	7.95	**99**	380.006	0.0304	0.0211
Random-250-3	250	956	73	257.891	opt	132.791	0.0305	0.0252
Random-250-4	250	1 194	62	1 406.04	opt	148.167	0.0224	0.0218
Random-250-30	250	9 347	13	1 408.412	**14**	1.005	0	0
Random-250-40	250	12 500	10	359.601	opt	0.743	0	0
Random-250-50	250	15 605	8	61.927	opt	0.621	0	0
Random-250-60	250	18 660	8	206.548	opt	0.037	0	0
Random-250-70	250	21 741	6	40.379	opt	0.037	0	0
Random-250-80	250	24 836	4	3.071	opt	0.465	0	0
Random-250-90	250	27 974	4	1.404	opt	0.052	0	0
Random-300-1	300	481	145	0.299	**149**	2 797.158	0.0221	0.0135
Random-300-2	300	876	103	116.818	**105**	1 057.238	0.0394	0.0192
Random-300-40	300	17 934	10	483.378	opt	3.232	0.0174	0.0437
Random-300-50	300	22 520	8	334.329	opt	31.909	0	0
Random-300-60	300	26 952	8	622.751	opt	0.069	0	0
Random-300-70	300	31 390	6	66.546	opt	0.286	0	0

Table 4 continues ...

Instance			CPLEX		VNS			
Name	$\vert V\vert$	$\vert E\vert$	opt	t_{cpl}	sol	t	*err*	σ
Random-300-80	300	35 871	5	34.579	opt	1.725	0.0667	0.0816
Random-300-90	300	40 412	4	2.191	opt	0.092	0	0

Table 4. Experimental results for the RD problem on random graph instances

Table 4 contains the results of the experimental testing on random generated graphs. As it can be seen, the VNS algorithm reaches the solution value equal to the optimal solution value on many instances (unsuccessful on 7 among 87 instances). On instances where an optimal solution was not reached, standard deviation σ is lower than 2.5 %. Instances "Random-200-8"–"Random-200-20", "Random-250-5"–"Random-250-20" and "Random-300-3"–"Random-300-30" are omitted from Table 4 because CPLEX was unable to find an optimal solution value on these instances. Nevertheless, the VNS algorithm finds some solution value for these instances, but because we do not have an optimal solution value on these instances, we will not present the VNS algorithm results either.

Before we present experimental results for the WRD problem on the same set of instances, let us summarize the results presented in Tables 1-4. The VNS algorithm for the RD problem finds solutions of good quality relatively fast, especially on instances of planar type. On instances of grid and net type, using CPLEX optimization solver is better, but on instances of planar and random type, using the VNS algorithm is preferable.

Experimental results of the VNS algorithm for the WRD problem are performed on instances where some solution values are known from the literature. Given that CPLEX was not able to solve the WRD problem on many instances within the time limit because of the "out of memory" status or because of the time limit, we tested the VNS algorithm both on instances where the optimal solution value is known and on instances where the found solution is not proved to be the optimal solution. Testings were made on 84 instances of different type. CPLEX optimization solver was able to find the optimal solution on 64 of them. The VNS algorithm was not able to find solutions equal to the optimal ones only on two instances. On instances where the optimal solution value is unknown, the VNS solutions are equal or better than the solutions found by CPLEX. Also, for almost all instances, the VNS algorithm runtime is lower than CPLEX runtime. Detailed information considering these testings is provided in Tables 5–8.

From Table 5 it can be concluded that the VNS reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful only on "grid06x13"). On instances where the optimal solution value is unknown, σ is lower than 2.2 %. Running times on instances where the optimal solution value is known shows that the VNS rapidly reaches these solutions in lower than 150 seconds. Even more, on many instances (38 of 42), running times are smaller than 30 seconds and only on "grid07x14" and "grid08x12" greater than 100 seconds. On instances where

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
grid04x10	40	66	15	15	4.109	opt	0.015	0	0
grid05x08	40	67	14	14	4.64	opt	0.047	0.0333	0.0333
grid03x14	42	67	16	16	4.829	opt	< 0.01	0	0
grid06x07	42	71	15	15	5.801	opt	0.08	0.0063	0.0188
grid04x11	44	73	16	16	5.5	opt	0.031	0.0088	0.0210
grid03x15	45	72	17	17	7.789	opt	0.012	0	0
grid05x09	45	76	16	16	7.908	opt	0.139	0.0235	0.0288
grid04x12	48	80	17	17	12.84	opt	0.069	0.0361	0.0265
grid06x08	48	82	18	18	25.499	opt	< 0.01	0	0
grid07x07	49	84	18	18	9.845	opt	0.021	0	0
grid05x10	50	85	18	18	10.61	opt	0.055	0.0053	0.0158
grid04x13	52	87	19	19	11.813	opt	0.035	0.0050	0.0150
grid06x09	54	93	19	19	25.539	opt	0.331	0.0450	0.0150
grid05x11	55	94	19	19	11.424	opt	0.388	0.0300	0.0245
grid04x14	56	94	20	20	35.326	opt	0.082	0.0214	0.0237
grid07x08	56	97	20	20	21.882	opt	0.076	0.0286	0.0233
grid04x15	60	101	22	22	40.256	opt	0.163	0	0
grid05x12	60	103	21	21	14.88	opt	4.036	0.0271	0.0260
grid06x10	60	104	21	21	35.713	opt	0.746	0.0273	0.0223
grid07x09	63	110	22	22	70.259	opt	0.318	0.0370	0.0155
grid08x08	64	112	23	23	171.925	opt	0.037	0.0063	0.0149
grid05x13	65	112	23	23	67.007	opt	0.928	0.0208	0.0208
grid06x11	66	115	24	24	381.771	opt	0.757	0.0040	0.0120
grid05x14	70	121	24	24	73.489	opt	27.03	0.0491	0.0202
grid07x10	70	123	25	25	618.089	opt	0.67	0.0077	0.0154
grid06x12	72	126	26	26	1 166.405	opt	0.544	0.0074	0.0148
grid08x09	72	127	25	25	435.146	opt	15.935	0.0383	0.0117
grid05x15	75	130	26	26	288.06	opt	8.133	0.0313	0.0174
grid07x11	77	136	27	27	988.596	opt	0.582	0.0268	0.0155
grid06x13	78	137	27	27	1 005.126	**28**	0.407	0.0086	0.0149
grid08x10	80	142	28	28	2 162.812	opt	10.011	0.0375	0.0178
grid09x09	81	144	28	28	737.579	opt	12.521	0.0437	0.0251
grid06x14	84	148	30	30	–	30	2.319	0.0097	0.0148
grid07x12	84	149	29	29	4 637.38	opt	47.642	0.0441	0.0181
grid08x11	88	157	31	31	–	31	3.412	0.0278	0.0190
grid06x15	90	159	32	32	–	32	1.196	0.0179	0.0218
grid09x10	90	161	31	31	–	31	40.651	0.0443	0.0197
grid07x13	91	162	32	32	–	32	16.778	0.0272	0.0130
grid08x12	96	172	33	33	–	33	107.765	0.0403	0.0184
grid07x14	98	175	34	34	1 720.86	opt	143.804	0.0433	0.0181
grid09x11	99	178	35	35	–	35	2.261	0.0181	0.0132
grid10x10	100	180	35	35	–	35	6.63	0.0302	0.0168

Table 5: Experimental results for the WRD problem on grid graph instances

optimization solvers were unable to prove optimality of the found solutions, the VNS heuristic reaches the same solution values for less than 108 seconds. So, we can conclude that the VNS heuristic solves the WRD problem on grid graph instance significantly faster than the optimization solver CPLEX and found solutions are of good quality.

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
plan10	10	27	3	3	0.156	opt	< 0.01	0	0
plan20	20	105	3	3	1.36	opt	< 0.01	0	0
plan30	30	182	5	5	7.49	opt	< 0.01	0	0
plan50	50	465	6	6	98.49	opt	0.01	0	0
plan100	100	1 540	9	9	–	$\underline{8}$	4.916	0	0
plan150	150	2 867	13	13	–	$\underline{10}$	88.248	0.0273	0.041

Table 6: Experimental results for the WRD problem on planar graph instances

From Table 6 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on all instances. Also, on instances where optimization solvers were unable to prove optimality of the found solution, the VNS solution is better. Again, running time for the instances where the optimal solution value is known is lower than 1 second. On "plan100", where optimization solvers were unable to prove optimality of the found solution, the proposed VNS algorithm finds solution value with σ equal to zero. On "plan150" the VNS solution is equal to 10 with $\sigma = 0.0417$, which can be considered as the solution of the good quality (solution value equal to 10 was reached in 14 of 20 runnings).

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
Net-10-10	100	342	20	20	148.213	opt	4.29	0.0095	0.0190
Net-10-20	200	712	40	40	–	40	67.323	0.0146	0.0119
Net-20-20	400	1 482	83	83	–	$\underline{81}$	2 066.577	0.0180	0.0132
Net-30-20	600	2 252	122	122	–	**123**	6 034.018	0.0474	0.0352

Table 7: Experimental results for the WRD problem on net graph instances

In Table 7 optimization solvers were able to find optimal solution value only for "Net-10-10". The same solution value was found by the proposed VNS algorithm with lower running time and with σ equal to 1.9 %. On "Net-10-20" and "Net-20-20" the VNS algorithm reaches the same and better solution value than optimization solvers, while for "Net-30-20" the VNS solution value is worse than the solvers' solution value.

From Table 8 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful only on 1 among 25 instances of random type). On instance "Random-100-6", where the

Instance				Solver		VNS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	opt	val	t	sol	t	*err*	σ
Random-50-1	50	49	24	24	0.281	opt	< 0.01	0	0
Random-50-2	50	49	23	23	0.343	opt	0.034	0	0
Random-50-3	50	58	24	24	0.39	opt	0.062	0	0
Random-50-4	50	54	24	24	0.484	opt	0.225	0	0
Random-50-5	50	67	22	22	0.968	opt	0.377	0.0196	0.0216
Random-50-6	50	86	19	19	2.053	opt	0.03	0	0
Random-50-7	50	84	19	19	3.171	opt	0.889	0.0175	0.0238
Random-50-8	50	95	17	17	3.093	opt	0.131	0.0333	0.0272
Random-50-9	50	108	17	17	26.373	opt	0.129	0.0028	0.0121
Random-50-10	50	112	16	16	6.781	opt	0.047	0	0
Random-50-20	50	248	9	9	346.264	opt	< 0.01	0	0
Random-50-30	50	373	7	7	476.278	opt	0.038	0	0
Random-50-40	50	475	6	6	1447.318	opt	0.092	0	0
Random-50-50	50	597	5	5	1545.06	opt	0.013	0	0
Random-50-60	50	739	4	4	210.71	opt	0.014	0	0
Random-50-70	50	860	3	3	156.14	opt	0.059	0	0
Random-50-80	50	980	3	3	90.813	opt	< 0.01	0	0
Random-50-90	50	1103	2	2	36.53	opt	0.03	0	0
Random-100-1	100	100	46	46	0.64	opt	157.329	0.0354	0.0145
Random-100-2	100	109	46	46	0.843	opt	36.052	0.0148	0.0117
Random-100-3	100	181	37	37	7.421	opt	23.64	0.0445	0.0261
Random-100-4	100	206	34	34	61.702	opt	12.367	0.0213	0.0175
Random-100-5	100	231	32	32	164.502	opt	60.361	0.0299	0.0186
Random-100-6	100	321	26	26	5 806.74	**27**	12.441	0.0265	0.0217
Random-100-7	100	317	25	25	4 009.377	opt	204.939	0.0434	0.0234
Random-100-8	100	317	23	23	–	23	313.924	0.0448	0.0279
Random-100-9	100	430	21	21	–	21	4.98	0.0269	0.0293
Random-100-10	100	498	19	19	–	19	460.905	0.0445	0.0260
Random-100-20	100	981	12	12	–	$\underline{11}$	8.951	0.0250	0.0382
Random-100-30	100	1 477	11	11	–	$\underline{8}$	1 462.462	0.1056	0.0242
Random-100-40	100	1 945	9	9	–	$\underline{7}$	1.501	0	0
Random-100-50	100	2 483	7	7	–	$\underline{5}$	37.134	0	0

Table 8: Experimental results for the WRD problem on random generated graph instances

optimal solution value was not reached, σ is equal to 2.17 %. Further, on instances "Random-100-40" and "Random-100-50", where optimization solvers were unable to prove optimality of the found solution, the VNS algorithm finds better solutions values with σ equal to zero for less than 38 seconds.

From Tables 5–8 we can see that optimization solvers were unable to provide an optimal solution value on instances of grid type with number of vertices larger than 84, on instances of planar and net type with number of vertices larger than

100 and on large number of instances of random type with 100 vertices. Also, we can see that, on the same set of instances, the VNS algorithm finds solutions of the WRD problem of good quality and, for many instances, faster than optimization solvers.

5 CONCLUSIONS

In this paper, the Variable Neighborhood Search approach for solving the Roman and the weak Roman domination problems is proposed. Tests were run on grid, net, planar and randomly generated graphs, with up to 600 vertices. The VNS was able to find solutions equal to the optimal ones for the RD problem on 218 of 231 tested instances and able to find solutions equal or better than CPLEX solutions for the WRD problem on 84 of 86 tested instances. Therefore, we can conclude that the VNS algorithm provides good quality solutions regardless of the type of instance and the type of problem, which makes it efficient for solving both the Roman and the weak Roman domination problems. Moreover, given the fact that optimization solvers were not able to solve the WRD problem on large scale instances (i.e., instances with more than 100 vertices) proposed algorithm can be used. Furthermore, given the fact that this algorithm does not contain any limitations on the number of variables and the number of conditions, it can be used for solving the RD problem on instances where optimization solvers are not able to provide an optimal solution value.

In future work, hybridization with some exact methods or application of some other heuristic could lead to possible better achievements in solving the Roman and the weak Roman domination problems.

Acknowledgments

This research has been partially supported by the Serbian Ministry of Education, Science and Technological Development under grants No. TR36006 and ON174010. The authors gratefully acknowledge the two referees of this paper for many excellent suggestions which have helped in improving this paper.

REFERENCES

[1] ReVelle, C. S.—Rosing, K. E.: Defendens Imperium Romanum: A Classical Problem in Military Strategy. The American Mathematical Monthly, Vol. 107, 2000, No. 7, pp. 585–594, doi: 10.2307/2589113.

[2] Cockayne, E. J.—Dreyer, P. A.—Hedetniemi, S. M.—Hedetniemi, S. T.: Roman Domination in Graphs. Discrete Mathematics, Vol. 278, 2004, No. 1-3, pp. 11–22, doi: 10.1016/j.disc.2003.06.004.

[3] Henning, M. A.—Hedetniemi, S. T.: Defending the Roman Empire – A New Strategy. Discrete Mathematics, Vol. 266, 2003, No. 1-3, pp. 239–251, doi: 10.1016/s0012-365x(02)00811-7.

[4] Burger, A. P.—de Villiers, A. P.—van Vuuren, J. H.: A Binary Programming Approach Towards Achieving Effective Graph Protection. Proceedings of the 2013 ORSSA Annual Conference, ORSSA, 2013, pp. 19–30.

[5] Ivanović, M.: Improved Mixed Integer Linear Programing Formulations for Roman Domination Problem. Publications de l'Institut Mathématique, Vol. 99, 2016, No. 113, pp. 51–58, doi: 10.2298/PIM1613051I.

[6] Ivanović, M.: Improved Integer Linear Programming Formulation for Weak Roman Domination Problem. Soft Computing, Vol. 22, 2018, No. 19, pp. 6583–6593, doi: 10.1007/s00500-017-2706-4.

[7] Liu, C. S.—Peng, S. L.—Tang, C. Y.: Weak Roman Domination on Block Graphs. Proceedings of the 27th Workshop on Combinatorial Mathematics and Computation Theory, Providence University, Taichung, Taiwan, April 30–May 1, 2010, pp. 86–89.

[8] Burger, A. P.—Cockayne, E. J.—Grundlingh, W. R.—Mynhardt, C. M.—Van Vuuren, J. H.—Winterbach, W.: Finite Order Domination in Graphs. Journal of Combinatorial Mathematics and Combinatorial Computing, Vol. 49, 2004, pp. 159–176.

[9] Stewart, I.: Defend the Roman Empire! Scientific American, Vol. 281, 1999, pp. 136–138, doi: 10.1038/scientificamerican1299-136.

[10] Chellali, M.—Haynes, T. W.—Hedetniemi, S. T.: Bounds on Weak Roman and 2-Rainbow Domination Numbers. Discrete Applied Mathematics, Vol. 178, 2014, pp. 27–32, doi: 10.1016/j.dam.2014.06.016.

[11] Dreyer Jr, P. A.: Applications and Variations of Domination in Graphs. Ph.D. thesis, Rutgers University, 2000.

[12] Cockayne, E. J.—Grobler, P. J. P.—Gründlingh, W. R.—Munganga, J.—van Vuuren, J. H.: Protection of a Graph. Utilitas Mathematica, Vol. 67, 2005, pp. 19–32.

[13] Currò, V.: The Roman Domination Problem on Grid Graphs. Ph.D. thesis, Università di Catania, 2014.

[14] Favaron, O.—Karami, H.—Khoeilar, R.—Sheikholeslami, S. M.: On the Roman Domination Number of a Graph. Discrete Mathematics, Vol. 309, 2009, No. 10, pp. 3447–3451, doi: 10.1016/j.disc.2008.09.043.

[15] Klobučar, A.—Puljić, I.: Some Results for Roman Domination Number on Cardinal Product of Paths and Cycles. Kragujevac Journal of Mathematics, Vol. 38, 2014, No. 1, pp. 83–94, doi: 10.5937/KgJMath1401083K.

[16] Klobučar, A.—Puljić, I.: Roman Domination Number on Cardinal Product of Paths and Cycles. Croatian Operational Research Review, Vol. 6, 2015, No. 1, pp. 71–78, doi: 10.17535/crorr.2015.0006.

[17] Xing, H.-M.—Chen, X.—Chen, X.-G.: A Note on Roman Domination in Graphs. Discrete Mathematics, Vol. 306, 2006, No. 24, pp. 3338–3340, doi: 10.1016/j.disc.2006.06.018.

[18] Wang, H.—Xu, X.—Yang, Y.—Ji, C.: Roman Domination Number of Generalized Petersen Graphs $p(n, 2)$. arXiv Preprint, arXiv:1103.2419, 2011.

[19] Pavlič, P.—Žerovnik, J.: Roman Domination Number of the Cartesian Products of Paths and Cycles. The Electronic Journal of Combinatorics, Vol. 19, 2012, No. 3, Art. No. P19.

[20] Liedloff, M.—Kloks, T.—Liu, J.—Peng, S.-L.: Efficient Algorithms for Roman Domination on Some Classes of Graphs. Discrete Applied Mathematics, Vol. 156, 2008, No. 18, pp. 3400–3415, doi: 10.1016/j.dam.2008.01.011.

[21] Liedloff, M.—Kloks, T.—Liu, J.—Peng, S. L.: Roman Domination over Some Graph Classes. In: Kratsch, D. (Ed.): Graph-Theoretic Concepts in Computer Science (WG 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3787, 2005, pp. 103–114, doi: 10.1007/11604686_10.

[22] Shang, W.—Hu, X.: The Roman Domination Problem in Unit Disk Graphs. In: Shi, Y., van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science (ICCS 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4489, 2007, pp. 305–312, doi: 10.1007/978-3-540-72588-6_51.

[23] Pushpam, P. R. L.—Malini Mai, T. N. M.: Weak Roman Domination in Graphs. Discussiones Mathematicae Graph Theory, Vol. 31, 2011, No. 1, pp. 161–170, doi: 10.7151/dmgt.1532.

[24] Lai, Y. L.—Lin, C. T.—Ho, H. M.: Weak Roman Domination on Graphs. Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory, National Penghu University of Science and Technology, Penghu, Taiwan, May 27–28, 2011, pp. 224–214.

[25] Peng, S.-L.—Tsai, Y.-H.: Roman Domination on Graphs of Bounded Treewidth. Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory, 2007, pp. 128–131.

[26] Chapelle, M.—Cochefert, M.—Couturier, J.-F.—Kratsch, D.—Liedloff, M.—Perez, A.: Exact Algorithms for Weak Roman Domination. In: Lecroq, T., Mouchard, L. (Eds.): Combinatorial Algorithms (IWOCA 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8288, 2013, pp. 81–93, doi: 10.1007/978-3-642-45278-9_8.

[27] Hansen, P.—Mladenović, N.—Urošević, D.: Variable Neighborhood Search for the Maximum Clique. Discrete Applied Mathematics, Vol. 145, 2004, No. 1, pp. 117–125, doi: 10.1016/j.dam.2003.09.012.

[28] Brimberg, J.—Mladenović, N.—Urošević, D.—Ngai, E.: Variable Neighborhood Search for the Heaviest k-Subgraph. Computers and Operations Research, Vol. 36, 2009, No. 11, pp. 2885–2891, doi: 10.1016/j.cor.2008.12.020.

[29] Mladenović, N.: A Variable Neighborhood Algorithm – A New Metaheuristic for Combinatorial Optimization. Papers Presented at Optimization Days, 1995, p. 112.

[30] Mladenović, N.—Hansen, P.: Variable Neighborhood Search. Computers and Operations Research, Vol. 24, 1997, No. 11, pp. 1097–1100, doi: 10.1016/s0305-0548(97)00031-2.

[31] Hansen, P.—Mladenović, N.: An Introduction to Variable Neighborhood Search. In: Voss, S., Martello, S., Osman, I. H., Roucairol, C. (Eds.): Meta-Heuristics. Springer, Boston, MA, 1999, pp. 433–458, doi: 10.1007/978-1-4615-5775-3_30.

Marija Ivanović finished her master studies at Faculty of Mathematics, University of Belgrade, in 2011. Since 2007 she has worked at the Faculty of Mathematics, Department for Numerical Mathematics and Optimization as Assistant. The main areas of research are game theory, combinatorial optimization and operations research. She participates in research projects financed by the Ministry of Education, Science and Technological Development, Serbia.

Dragan Urošević finished his Ph.D. studies at Faculty of Mathematics, University of Belgrade, in 2004. Since 1993 he has worked at the Mathematical Institute SANU. The main areas of research are combinatorial optimization and operations research. He is engaged in the development and implementation of heuristic methods to solve complex problems in graph theory and the development of methods for solving location problems. He participates in research projects financed by the Ministry of Education, Science and Technological Development, Serbia.

A Mixed Integer Linear Programming Formulation for Restrained Roman Domination Problem

Marija Ivanovic
Institute of Physics Belgrade
7 PUBLICATIONS **25** CITATIONS

Theory and Applications of Mathematics & Computer Science

(ISSN 2067-2764, EISSN 2247-6202)
http://www.uav.ro/applications/se/journal/index.php/tamcs

Theory and Applications of Mathematics & Computer Science 5 (2) (2015) 110– 115

A Mixed Integer Linear Programming Formulation for Restrained Roman Domination Problem

Marija Ivanović[a]

[a]*Faculty of Mathematics, University of Belgrade, Studentski trg 16/IV, 11 000 Belgrade, Serbia*

Abstract

This paper deals with a subgroup of Roman domination problems (RDP) named Restrained Roman domination problem (RRDP). It introduces a new mixed integer linear programming (MILP) formulation for the RRDP. The presented model uses relatively small number of the variables and constraints and could be of use both in theoretical and practical purposes. Proof of its correctness is given, i.e. it was shown that optimal solution to the RRDP formulation is equal to the optimal solution of the original problem.

Keywords: Restrained Roman domination in graphs, combinatorial optimization, integer linear programming.
2010 MSC: 90C11, 05C69.

1. Introduction

With contiguous territories throughout Europe, North Africa, and the Middle East, the Roman Empire was one of the largest in history (Kelly, 2006). The idea of building ”empire without end” (Nicolet, 1991) expressed the ideology that neither time nor space limited the Empire. During the fourth century A.D., Emperor of Rome, Constantine the Great, intended to accomplish that idea. In order to expand the Roman Empire, he dealt with the next problem: How to organize legions such that entire Empire of Rome stayed defended? Since legions were highly trained, it was assumed that they could move fast from one city to another. City was considered to be defended if at least one legion was stationed in it or it was adjacent to a city with two legions within. The second condition was made because legion could move from a stationed city only if such an act won't leave it undefended.

Inspired by this historical problem, a new subgroup of the domination problems, named Roman domination problem (RDP), was proposed by Stewart (1999). RDP can be described as a problem of finding the minimal number of legions such that entire Empire of Rome is defended.

Email address: `marijai@math.rs` (Marija Ivanović)

More details about the RDP can be found in (ReVelle & Rosing, 2000), (Currò, 2014), (Liedloff *et al.*, 2005) and (Xing *et al.*, 2006).

Restrained Roman domination problem (RRDP), previously introduced by Pushpam & Sampath (2015), is defined also as a problem of finding the minimal number of legions such that entire Empire of Rome is defended but the conditions are slightly changed. Again, a city is considered to be defended if at least one legion is stationed within. But, a city without legion within is consider to be defended if it is adjacent to at least one city with two legions within and to at least one undefended city.

The Roman domination problem and the Restrained Roman domination problem can be illustrated by a graph such that each city of the Empire of Rome is represented by a vertex and, for two connected cities, the corresponding vertices are set to be adjacent.

Assuming that five cities, marked by numbers 1 - 5, are constructed such that a city marked by 1 is only adjacent to a city marked by 2 and that all other cities are adjacent to each other, a small illustration of the RDP and RRDP solutions are given in the figure below.

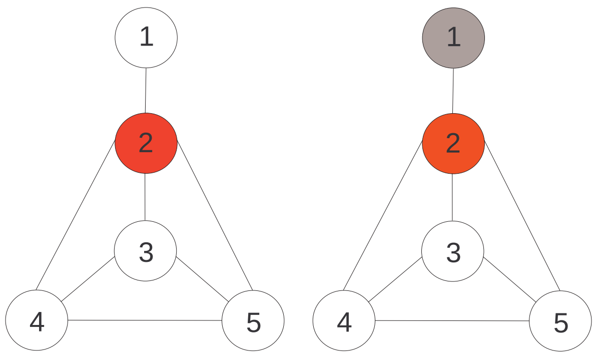


Figure 1. Illustrations of the the RDP (left) and RRDP (right) solutions

Vertex colored in red indicates that corresponding city is defended by two legions, vertex colored in gray indicates that corresponding city is defended by one legion, while vertices colored in white stands for the cities without legions within. For the Roman domination problem (shown on the figure on the left) by using only two legions, all five cities could be defended, i.e. assigning two legions to a city marked by 2, corresponding city and all adjacent cities are consider to be defended. Note that minimal number of legions for the RRDP (shown on the figure on the right) is three, i.e. assigning two legions to a city marked by 2, corresponding city and cities marked by 3, 4 and 5 are set to be defended because they are adjacent to a city with two legions within and adjacent to two cities with no assigned legions; city marked by 1 is set to be defended by one legion, since it can't be adjacent to at least one city without legions within and to at least one city with two legions within, at the same time. Given solution for RRDP is not unique since the same result could be obtained by assigning two legions to a city marked by 3 instead of the city marked by 2.

In the next sections, MILP formulation for the RRDP together with the proof of its validity, are proposed.

2. Problem definition

Let $G = (V, E)$ be an undirected graph with a vertex set V such that each vertex $u \in V$ represents a city of Roman Empire and each edge, $e \in E$, represents an existing road between two adjacent cities. A neighborhood set N_u ($N_u \subset V$), of a vertex $u \in V$, is defined as a set of vertices v adjacent to a vertex u. For a function f

$$f : V \to \{0, 1, 2\} \tag{2.1}$$

let a number of legions assigned to a city represented by a vertex u to be equal to a value $f(u)$. Additionally, let a function f satisfy the condition that for every vertex $u \in V$such that $f(u) = 0$ there exists vertices $v, w \in V$ such that $f(v) = 2$ and $f(w) = 0$. In other words, if there is an undefended city u, then there exist at least one city $v, v \in N_u$ with two legions within and at least one undefended city $w, w \in N_u$. Function f is called a restrained Roman domination function.

Mathematically, a proposed problem can be formulated as:

$$\min_f F_1(f) \tag{2.2}$$

subject to:

$$F_1(f) = \sum_{u \in V} f(u) \tag{2.3}$$

$$(\forall u \in V) f(u) = 0 \Rightarrow (\exists v, w \in N_u)(f(v) = 2 \wedge f(w) = 0). \tag{2.4}$$

Now, using a proposed notations, a solution to the illustrated RRDP can be written as: $F_1(f) = 3$ for $f(2) = 2$, $f(1) = 1$ and $f(3) = f(4) = f(5) = 0$ and it is not unique ($F_1(f) = 3$ for $f(3) = 2$, $f(1) = 1$ and $f(2) = f(4) = f(5) = 0$).

3. A mixed integer linear programming formulation for the RRDP

For a function f, defined by (2.1), let a continuous decision variable x_i, $x_i \in [0, \infty)$, indicate a number of legions assigned to a corresponding city $i \in V$. Although, $f \in \{0, 1, 2\}$ and $x_i \in [0, \infty)$, x_i and $f(i)$ are with equal values in the optimal solution, and not necessary with equal values for every feasible solution. Let binary decision variables y_i and z_i indicate if there are two or none legions assigned to a corresponding city $i \in V$,

$$y_i = \begin{cases} 1, & f(i) = 2 \\ 0, & otherwise \end{cases} \quad \text{and} \quad z_i = \begin{cases} 1, & f(i) = 0 \\ 0, & otherwise \end{cases}.$$

A mixed integer linear programming (MILP) formulation for the RRDP can now be formulated as follows:

$$\min \sum_{i \in V} x_i \tag{3.1}$$

subject to

$$x_i + \sum_{j \in N_i} y_j \geq 1, \qquad i \in V \tag{3.2}$$

$$x_i + \sum_{j \in N_i} z_j \geq 1, \qquad i \in V \tag{3.3}$$

$$x_i \geq 2y_i, \qquad i \in V \tag{3.4}$$

$$x_i + 2z_i \leq 2, \qquad i \in V \tag{3.5}$$

$$x_i \in [0, +\infty); \qquad y_i, z_i \in \{0, 1\}, \quad i \in V. \tag{3.6}$$

Further, for vector values $x = [x_i], y = [y_i]$ and $z = [z_i]$, which satisfies constraints (3.2) - (3.6), notation $F_2(x, y, z) = \sum_{i \in V} x_i$ will be used. Now, condition (3.1) which minimizes the number of legions, can be written as $\min\limits_{(x,y,z)} F_2(x, y, z)$. By the constraints (3.2) it is ensured that each undefended vertex i is adjacent to at least one vertex with two legions within. Similarly, by the constraints (3.3) it is ensured that each undefended vertex i is adjacent to at least one vertex which is also undefended. From the inequalities (3.4) and (3.5) it follows that for each city $i \in V$ with at most 1 legion within, corresponding value y_i is set to be equal to zero and that for each city $i \in V$ with at least one legions within, corresponding value z_i is set to be equal to zero. Finally, decision variables x are set to be continuous, while y and z are set to be binary by the constraints (3.6).

A given MILP formulation consists of $2|V|$ variables which are binary and $|V|$ continuous variables. Number of constraints is equal to $4|V|$.

A proof of the validity of the MILP formulation for the RRDP is given in the next proposition.

Proposition 1. *The optimal objective function value $F_1(f)$ of the Restrained Roman domination problem (2.1) - (2.4) is equal to the optimal objective function value $F_2(x, y, z)$ of the MILP formulation (3.1) - (3.6).*

Proof. ($\Rightarrow$) In this part will be proven that the optimal objective function value of the Restrained Roman domination problem (2.1) - (2.4) is greater or equal to the optimal objective function value of the MILP formulation (3.1) - (3.6), i.e. $F_1(f) \geq F_2(x, y, z)$.

For a fixed city $i \in V$ and a function f given by (2.1), let decision variables x_i, y_i and z_i be defined as

$$x_i = f(i), \qquad y_i = \begin{cases} 1, & f(i) = 2 \\ 0, & otherwise \end{cases} \quad \text{and} \quad z_i = \begin{cases} 1, & f(i) = 0 \\ 0, & otherwise \end{cases}.$$

Since $x_i \in [0, +\infty)$, $y_i, z_i \in \{0, 1\}$, conditions (3.4) - (3.6) are satisfied by the definition. For example, a condition (3.5) is satisfied because $z_i = 1$ for $x_i = f(i) = 0$ ($x_i + 2z_i = 2$) and $z_i = 0$ for

$x_i = f(i) = 1$ ($x_i + 2z_i = 1 < 2$). Similarly $z_i = 0$ for $x_i = f(i) = 2$, which again implies that $x_i + 2z_i = 2$.

Assuming that conditions (3.2) and (3.3) holds for a fixed vertex $i \in V$, there are two cases:
Case 1. Let values $f(i)$ are set to be greater or equal to one. Since $x_i = f(i)$, relation $x_i \geq 1$ implies. From the last relation and by the binary notations of the variables y_i and z_i it implies that $x_i + \sum_{j \in N_i} y_j \geq 1$ and $x_i + \sum_{j \in N_i} z_j \geq 1$.
Case 2. Let values $f(i)$ are set to be equal to zero. Satisfying relation (2.4) $(\exists v, w \in N_u)(f(v) = 2 \wedge f(w) = 0)$, it follows that $y_v = 1$ and $z_w = 1$. Therefore, $x_i + \sum_{j \in N_i} y_j = \sum_{j \in N_i} y_j \geq 1$ and $x_i + \sum_{j \in N_i} z_j = \sum_{j \in N_i} z_j \geq 1$.

Finally, since decision variables satisfies the conditions (3.1) - (3.6) for a fixed vertex i, it follows that $F_2(x, y, z) = \sum_{i \in V} x_i = \sum_{i \in V} f(i) = F_1(f)$.

($\Leftarrow$) In this part it will be proven that optimal objective function value of the Restrained Roman domination problem (2.1) - (2.4) is less or equal to the optimal objective function value of the MILP formulation (3.1) - (3.6), i.e. $F_1(f) \leq F_2(x, y, z)$.

For a given set of decision variables x_i, y_i and z_i which satisfy conditions (3.1) - (3.6), let a function f be defined as

$$f(i) = \begin{cases} 0, & x_i \in [0, 1) \\ 1, & x_i \in [1, 2) \\ 2, & x_i \in [2, +\infty) \end{cases} . \tag{3.7}$$

By the definition of the function f, condition (2.1) holds. Since the condition (2.1) holds, for a fixed vertex $u \in V$ there are two cases:
Case 1. Let $x_u \in [1, +\infty)$. By the definition of the function f, it follows that $f(u) = 1$ or $f(u) = 2$. Now, condition (2.4) holds, since $\perp \Rightarrow p$ is tautology for any logical statement p.
Case 2. Let $x_u \in [0, 1)$. By the definition of the function f, $f(u) = 0$. Because of the condition (3.2), $x_u + \sum_{j \in N_u} y_j \geq 1$, it follows that $\sum_{j \in N_u} y_j \geq 1 - x_u > 0$. Since the decision variables y_j are binary, $\sum_{j \in N_u} y_j$ has to be integer, which implies that $\sum_{j \in N_u} y_j \geq 1$. Therefore, there exists a vertex $v \in N_u$, $y_v = 1$. From the constraints (3.4), and because of the $x_v \geq 2y_v = 2$, it follows that $f(v) = 2$. Similarly, from the constraints (3.3) it follows that $\sum_{j \in N_u} z_j \geq 1 - x_u > 0$. Because of the binary type of the decision variables z_j, $\sum_{j \in N_u} z_j$ has an integer value. Now, since $\sum_{j \in N_u} z_j \geq 1$, there exists a vertex $w \in N_u$ such that $z_w = 1$. Finally, by the constraints (3.5), $x_w \leq 2 - 2z_w = 0$, it follows that $x_w = 0$ and that $f(w) = 0$ which means that condition (2.4) holds also.

By the definition of the function f, it is clear that $f(i) \leq x_i$, for $i \in V$. Therefore, $F_1(f) = \sum_{i \in V} f(i) \leq \sum_{i \in V} x_i = F_2(x, y, z)$.

So, for each feasible solution to the problem (2.1) - (2.4) there exists a feasible solution to the problem (3.1) - (3.6), satisfying the relation $F_2(x, y, z) \leq F_1(f)$, and for each feasible solution to the (3.1) - (3.6) there exists a feasible solution to the (2.1) - (2.4) satisfying the relation $F_1(f) \leq$

$F_2(x, y, z)$. Therefore, it follows that $\min\limits_{f} F_1(f) = \min\limits_{(x,y,z)} F_2(x, y, z)$. $\square$

Applying the given MILP formulation to the illustrated RRDP, solution $\min\limits_{(x,y,z)} F_2(x, y, z)$ to the proposed problem is equal to 3, and it can be obtained for $x = [1, 0, 2, 0, 0]$, $y = [0, 1, 0, 0, 0]$ and $z = [0, 0, 1, 1, 1]$.

4. Conclusions

This paper is devoted to the Restrained Roman domination problem. A mixed integer linear programming formulation is introduced and the correctness of the corresponding formulation is proved. The presented model uses relatively small number of the variables and constraints, which indicates that presented model can be used both in theoretical and practical considerations. As a future study, it is planned to construct an exact method for solving the corresponding mathematical model. Construction of the metaheuristics for solving the proposed problem can also be a part of a possible future study.

Acknowledgments This research has been supported by the Research Grants 174010 and TR36015 of the Serbia Ministry of Education, Science and Technological Developments.

References

Currò, Vincenzo (2014). The Roman Domination Problem on Grid Graphs. PhD thesis. Università di Catania.

Kelly, Christopher (2006). *The Roman Empire: A Very Short Introduction*. Oxford University Press.

Liedloff, Mathieu, Ton Kloks, Jiping Liu and Sheng-Lung Peng (2005). Roman domination over some graph classes. In: *Graph-Theoretic Concepts in Computer Science*. Springer. pp. 103–114.

Nicolet, Claude (1991). *Space, Geography, and Politics in the Early Roman Empire*. University of Michigan Press.

Pushpam, Roushini Leely and Padmapriea Sampath (2015). Restrained roman domination in graphs. *Transactions on Combinatorics* **4**(1), 1–17.

ReVelle, Charles S and Kenneth E Rosing (2000). Defendens imperium romanum: a classical problem in military strategy. *American Mathematical Monthly* pp. 585–594.

Stewart, Ian (1999). Defend the roman empire!. *Scientific American* **281**, 136–138.

Xing, Hua-Ming, Xin Chen and Xue-Gang Chen (2006). A note on roman domination in graphs. *Discrete mathematics* **306**(24), 3338–3340.

PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 99(113) (2016), 51–58 DOI: 10.2298/PIM1613051I

IMPROVED MIXED INTEGER LINEAR PROGRAMING FORMULATIONS FOR ROMAN DOMINATION PROBLEM

Marija Ivanović

Abstract. The Roman domination problem is considered. An improvement of two existing Integer Linear Programing (ILP) formulations is proposed and a comparison between the old and new ones is given. Correctness proofs show that improved linear programing formulations are equivalent to the existing ones regardless of the variables relaxation and usage of lesser number of constraints.

1. Introduction

Domination on a graph has been extensively studied in the literature. Many variations and generalizations of this problem arose. One of them, with historical significants, was called the Roman domination problem. The Roman domination problem dates from the 4th century, when the Emperor of Rome, Constantine the Great, in order to defend an entire Empire, decreed that two types of legions should be placed in Roman provinces. The first type of legion were highly trained mobile warriors and therefore, in order to defend a province against any attack, they could move fast from one province to another. The second type of legion behaved as a local militia and were permanently stationed in a given province. The Emperor decreed that no mobile legion could ever leave a province in order to defend another one, if such an act would leave an originating province undefended.

The Roman Empire can be illustrated by an undirected graph where each province is represented by a different vertex. Two vertices will be set as adjacent if a connection between corresponding provinces exists. For connected provinces it will be said that they are neighbors. Any legion could move only over connected provinces, i.e., legion could move only along the edge of two adjacent vertices. Further, a province will be considered as defended if there is at least one legion

2010 *Mathematics Subject Classification:* 65K05, 90C11, 90C05, 94C15, 68R10.

Key words and phrases: Roman domination in graphs, combinatorial optimization, mixed integer linear programming.

Partially supported by the Serbian Ministry of Education, Science and Technological Developments, grant TR36006.

Communicated by Slobodan K. Simić.

stationed within it. A province without a stationed legion will be considered to be defended if a vertex which represents it is directly connected to a vertex which represents a province with two stationed legions: if there are two legions in a neighbor's province, one legion will be considered as mobile and therefore, a certain province will be considered as defended because a mobile legion could arrive fast in order to defend it. Otherwise a province is left to be undefended. For a detailed illustration see [**8**, p. 586].

The proposed problem is illustrated on a small example where the Emperor of Rome, Constantine the Great, had at his disposal only four legions to be placed and eight provinces to be defended, see Figure 1.

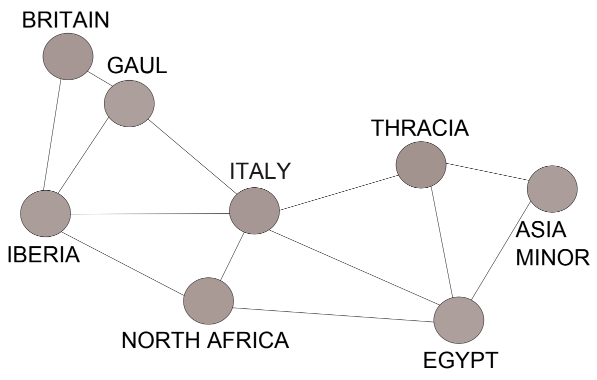


Figure 1. Representation of the Roman Empire illustrated on a graph.

Assigning two legions to Italy and another two to Thracia, one province was left to be undefended. Given that, in order to defend Britain, one mobile legion should move from Italy to Gaul waiting for another mobile legion to come from Thracia and then proceed to Britain. It is obvious that such strategy is not optimal, i.e. by assigning two legions to Iberia and another two to Egypt, the entire Empire will be defended. The same result could be reached also by assigning two legions to Britain or to Gaul and another two to Egypt. Note that the minimal number of legions necessary to defend the given Empire of Rome is four.

The Roman domination problem (RDP), introduced by Ian Stewart in [**10**], can be described as a problem of finding the minimal number of legions such that the entire Empire of Rome is safe.

There are several papers on this problem. The first additional developments of the RDP were proposed by ReVelle and Rosing in [**8**], while some of the most recent theoretical developments were given in [**1**, **3**, **5**–**7**].

Some special classes of graphs, such as interval graphs, intersection graphs, co-graphs and distance-hereditary graphs can be solved in linear time [**5**] but, in a general case, the Roman domination problem is NP-hard, [**4**, **5**, **9**].

This paper is organized as follows: the definition of the Roman domination problem is given in Section 2 and in Section 3, as proposed in the literature, two existing ILP formulations for solving the Roman domination problem will be reviewed. Subsequently, new alternative formulations together with the proofs of their correctness will be presented in Section 4. Conclusion, outlook on the future work and literature are given in the two final sections.

2. Problem definition

Let $G = (V, E)$ represents a finite and undirected graph with a vertex set V such that each vertex $v \in V$ represents a province and each edge, $e \in E$, represents an existing road between two adjacent provinces. There will be no loops nor multiple edges between two adjacent vertices. Let us define a neighborhood set N_v ($N_v \subset V$), of a vertex $v \in V$, such that each vertex $w \in N_v$ is adjacent to a vertex v. Finally, let us define a function $f : V \to \{0, 1, 2\}$ such that $f(v)$ is equal to a number of legions assigned to a province represented by a vertex v. Function f has to satisfy the condition that for every vertex $v \in V$ such that $f(v) = 0$ there exists a vertex $w \in N_v$ such that $f(w) = 2$. In other words, if there is an undefended province v, then there exists at least one province w, $w \in N_v$ with two stationed legions.

A small illustration of the Roman domination problem follows.

Example 2.1. Let as assume that there are 25 provinces to be defended and that each province can be represented by a particular vertex of a grid graph $G_{5,5}$. An adjacency matrix of a given graph is defined in such a way that it reflects a connection between the provinces illustrated on a Figure 2 (left).

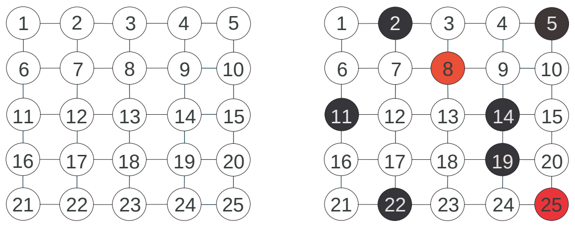


Figure 2. Illustration of Example 1.

The solution to the given problem is illustrated by coloring vertices in three colors, see Figure 2 (right) and is obtained by mathematical formulations described in Section 3. The vertices colored in black represent provinces with two assigned legions, colored in red represent provinces with one legion assigned and colored in white otherwise. Note that the entire area of 25 provinces could be defended by 14 legions, i.e. $f(v_2^*) = 2$ for all $v_2^* \in \{2, 5, 11, 14, 19, 22\}$, $f(v_1^*) = 1$ for all $v_1^* \in \{8, 25\}$ and $f(v_0^*) = 0$ for all $v_0^* \in \{1, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 17, 18, 20, 21, 23, 24\}$.

3. Existing integer linear programing formulations

sec1 There are two ILP formulations known from the literature. The first formulation was introduced by ReVelle and Rosing in [**8**] and will be referred to as

$\mathcal{RR}$, while the second formulation was introduced by Burger at el. in [**2**] and will be referred to as $\mathcal{BVV}$.

3.1. $\mathcal{RR}$ formulation. For a function f, $f : V \to \{0,1,2\}$, and $i \in V$, let us define the variables

$$x_i = \begin{cases} 1, & f(i) \geqslant 1 \\ 0, & \text{otherwise} \end{cases} \qquad y_i = \begin{cases} 1, & f(i) = 2 \\ 0, & \text{otherwise.} \end{cases}$$

The $\mathcal{RR}$ formulation of the RDP can be described by

$$\min \sum_{i \in V} x_i + \sum_{i \in V} y_i \tag{3.1}$$

$$x_i + \sum_{j \in N_i} y_j \geqslant 1, \quad i \in V \tag{3.2}$$

$$y_i \leqslant x_i, \quad i \in V \tag{3.3}$$

$$x_i, y_i \in \{0,1\}, \quad i \in V \tag{3.4}$$

The objective function value, given by (3.1), represents the number of legions used in defense. Constraints (3.2) ensure that each province is safe or there is at least one province in its neighborhood with two legions within it. By the constraints (3.3) it is ensured that provinces with two legions are safe. Decision variables x_i and y_i are preserved to be binary by the constraints (3.4).

The $\mathcal{RR}$ formulation consists of $2|V|$ binary variables and $2|V|$ constraints.

3.2. $\mathcal{BVV}$ formulation. For a function f, $f : V \to \{0,1,2\}$, let

$$x_i = \begin{cases} 1, & f(i) = 1 \\ 0, & \text{otherwise} \end{cases} \qquad y_i = \begin{cases} 1, & f(i) = 2 \\ 0, & \text{otherwise.} \end{cases}$$

The $\mathcal{BVV}$ formulation of the RDP can now be described by

$$\min \sum_{i \in V} x_i + 2 \sum_{i \in V} y_i \tag{3.5}$$

$$x_i + y_i + \sum_{j \in N_i} y_j \geqslant 1, \qquad i \in V \tag{3.6}$$

$$x_i + y_i \leqslant 1, \qquad i \in V \tag{3.7}$$

$$x_i, y_i \in \{0,1\}, \quad i \in V \tag{3.8}$$

The objective function value is given by (3.5). By conditions (3.6) it is obtained that an undefended province has to be in the neighborhood of at least one province with two assigned legions. By conditions (3.7) it is given that if a province is defended by two legions then there is no need to say that it is defended by one legion as well and vice versa. Again, the decision variables x_i and y_i are preserved to be binary by the constraints (3.8).

The $\mathcal{BVV}$ formulation also consists of $2|V|$ binary variables and $2|V|$ constraints.

4. Alternative linear programing formulations

4.1. New improved $\mathcal{RR}$ formulation. Let us define

$$x_i \in [0, +\infty), y_i \in \{0, 1\}, \qquad i \in V. \tag{4.1}$$

Considering the $\mathcal{RR}$ formulation it can be noted that the binary variables x_i can be relaxed to be real. Let us mark the $\mathcal{RR}$ formulation with the given relaxation (4.1) as $\mathcal{RR}_{Imp}$. Given that, the existing $\mathcal{RR}$ formulation, which is ILP formulation, can be relaxed to the MILP $\mathcal{RR}_{Imp}$ formulation.

Theorem 4.1. *Optimal objective function value of the $\mathcal{RR}$ formulation* (3.1)–(3.4)*, is equal to the optimal objective function value of the $\mathcal{RR}_{Imp}$ formulation* (3.1)–(3.3)*,* (4.1)*.*

Proof. Let a feasible solution to the $\mathcal{RR}_{Imp}$ formulation be represented by a vector $(\bar{x}'', \bar{y}'')$ where $\bar{x}'' = (x_1'', \dots, x_n'')$ and $\bar{y}'' = (y_1'', \dots, y_n'')$, $n = |V|$. Given that, let a vector $(\bar{x}', \bar{y}')$ $(\bar{x}' = (x_1', \dots, x_n')$, $\bar{y}' = (y_1', \dots, y_n'))$ of variables x_i' and y_i' be defined such that $y_i' = y_i''$ for each $i \in V$ and

$$x_i' = \begin{cases} 0, & x_i'' \in [0, 1) \\ 1, & x_i'' \in [1, +\infty). \end{cases}$$

By this definition, the variables x_i' and y_i' have binary values and therefore satisfy conditions (3.4). Combining the definitions of variables x_i'' and binary notation of variables y_i'', it follows that $y_i' = y_i'' \leqslant x_i''$. Now, if relations $y_i'' = 1$ stand, then inequalities $1 \leqslant x_i''$ imply that $x_i' = 1 \geqslant 1 = y_i'$. More, relations $y_i'' = 0$ provide inequalities $0 \leqslant x_i''$, which imply that $x_i' \in \{0, 1\} \geqslant 0 = y_i'$ for each $i \in V$. Therefore, conditions (3.3) are satisfied.
Assuming that $x_i'' + \sum_{j \in N_i} y_j'' \geqslant 1$, two cases arise:

1) $(\exists j \in N_i)$ such that $y_j'' = 1$,
2) $(\forall j \in N_i)$ $y_j'' = 0$.

The first case implies that there exists j such that $y_j' = 1$, i.e., $\sum_{j \in N_i} y_j' \geqslant 1$. Therefore, $x_i' + \sum_{j \in N_i} y_j' \geqslant 1$. The second case implies that $1 \leqslant x_i'' + \sum_{j \in N_i} y_j'' = x_i''$. Now, because of $x_i'' \geqslant 1$, it follows that $x_i' = 1$. Given that, conditions (3.2) are satisfied.

Finally, the objective function value of the $\mathcal{RR}$ formulation can be calculated as $\sum_{i \in V} x_i' + \sum_{i \in V} y_i'$, but because of the relations $y_i' = y_i''$ and $x_i' \leqslant x_i''$ it is easy to notice that $\mathrm{Obj}_{\mathcal{RR}} \leqslant \mathrm{Obj}_{\mathcal{RR}_{Imp}}$.

The objective function value of every relaxed minimization problem is less than or equal to the objective function value of the associated original problem, i.e. $\mathrm{Obj}_{\mathcal{RR}_{Imp}} \leqslant \mathrm{Obj}_{\mathcal{RR}}$. Finally, combining the given inequalities the theorem is proven, i.e., $\mathrm{Obj}_{\mathcal{RR}} = \mathrm{Obj}_{\mathcal{RR}_{Imp}}$. □

From the above it can be noted that $2|V|$ binary variables of the existing $\mathcal{RR}$ formulation can be replaced with $|V|$ binary and $|V|$ real variables without losing generality.

4.2. New improved $\mathcal{BVV}$ formulations. Considering the $\mathcal{BVV}$ formulation it can be noted that conditions (3.7) can be omitted. Formulation (3.5), (3.6) and (3.8) will be marked as $\mathcal{BVV}_{\text{Imp1}}$.

Moreover, following the idea for improving the $\mathcal{RR}$ formulation, the $\mathcal{BVV}_{\text{Imp1}}$ formulation can be further improved by relaxing binary variables x_i to be real. Such an improvement, (3.5), (3.6) and (4.1), will be marked as $\mathcal{BVV}_{\text{Imp2}}$.

Theorem 4.2. *Optimal objective function value of the $\mathcal{BVV}_{\text{Imp1}}$ formulation* (3.5)*,* (3.6) *and* (3.8) *is equal to the optimal objective function value of the $\mathcal{BVV}$ formulation* (3.5)*–*(3.8)*.*

Proof. Let a feasible solution to the $\mathcal{BVV}_{\text{Imp1}}$ formulation be represented by a vector $(\bar{x}'', \bar{y}'')$ where $\bar{x}'' = (x_1'', \dots, x_n'')$, $\bar{y}'' = (y_1'', \dots, y_n'')$, $n = |V|$. Now, let us define a vector $(\bar{x}', \bar{y}')$ of variables $\bar{x}' = (x_1', \dots, x_n')$, $\bar{y}' = (y_1', \dots, y_n')$ such that $y_i' = y_i''$ for each $i \in V$. Given that, we can define two disjunctive sets V_1 and V_2 such that $V_1 \cup V_2 = V$:

1) $x_i'' = 0$ or $y_i'' = 0$, for each vertex $i \in V_1$,
2) $x_i'' = 1$ and $y_i'' = 1$, for each vertex $i \in V_2$.

For each $i \in V_1$, let us define x_i' such that $x_i' = x_i''$. By the definition of the variables x_i'' and y_i'' and because $(x_i', y_i') = (x_i'', y_i'')$, conditions (3.6) and (3.8) are satisfied. Further, for a given set V_1 and because of the binary notations of the variables x_i'' and y_i'' conditions (3.7) are also satisfied:

$$x_i' + y_i' = x_i'' + y_i'' \leqslant \max\{x_i'', y_i''\} \in \{0,1\} \leqslant 1.$$

Now, let us define x_i' for each $i \in V_2$. Because of the definition of the variables y_i' and set V_2 ($y_i' = y_i''$ and $y_i'' = 1$), variables x_i' will be set to be equal to zero, i.e. $x_i' = 0$.

Conditions (3.6)–(3.8) are satisfied again:

$$x_i' + y_i' + \sum_{j \in N_i} y_i' = 0 + 1 + \sum_{j \in N_i} y_i'' = 1 + |V_2| \geqslant 1, \qquad (\forall i) \in V_2$$
$$x_i' + y_i' = 0 + 1 \leqslant 1, \qquad (\forall i) \in V_2$$
$$x_i' = 0 \in \{0,1\}, \qquad y_i' = 1 \in \{0,1\}$$

Therefore, a feasible solution to the $\mathcal{BVV}_{\text{Imp1}}$ formulation is a feasible solution to the $\mathcal{BVV}$ formulation. The objective function value can be calculated as follows:

$$\begin{aligned} \text{Obj}_{\mathcal{BVV}_{\text{Imp1}}} &= \sum_{i \in V} x_i'' + 2\sum_{i \in V} y_i'' = \sum_{i \in V_1} x_i'' + 2\sum_{i \in V_1} y_i'' + \sum_{i \in V_2} x_i'' + 2\sum_{i \in V_2} y_i'' \\ &= \sum_{i \in V_1} x_i'' + 2\sum_{i \in V_1} y_i'' + 3|V_2|. \end{aligned}$$

$$\begin{aligned} \text{Obj}_{\mathcal{BVV}} &= \sum_{i \in V} x_i' + 2\sum_{i \in V} y_i' = \sum_{i \in V_1} x_i' + 2\sum_{i \in V_1} y_i' + \sum_{i \in V_2} x_i' + 2\sum_{i \in V_2} y_i' \\ &= \sum_{i \in V_1} x_i'' + 2\sum_{i \in V_1} y_i'' + 2|V_2|. \end{aligned}$$

It is easy to notice that $\mathrm{Obj}_{\mathcal{BVV}} \leqslant \mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}}$.

Now, let us assume that U is a solution set to the $\mathcal{BVV}$ formulation. Omitting condition (3.7) from the $\mathcal{BVV}$ formulation, the solution to the $\mathcal{BVV}_{\mathrm{Imp1}}$ formulation can be marked as U_1. It is obvious that $U \subseteq U_1$ and therefore, a feasible solution to the $\mathcal{BVV}$ formulation is also a feasible solution to the $\mathcal{BVV}_{\mathrm{Imp1}}$ formulation. By the definition of the global and local minimums it implies that

$$\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} = \min_{U_1}\Big(\sum_{i\in V} x_i + 2\sum_{i\in V} y_i\Big) \leqslant \min_{U}\Big(\sum_{i\in V} x_i + 2\sum_{i\in V} y_i\Big) = \mathrm{Obj}_{\mathcal{BVV}}$$

Finally, combining $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} \geqslant \mathrm{Obj}_{\mathcal{BVV}}$ and $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} \leqslant \mathrm{Obj}_{\mathcal{BVV}}$ it implies that $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} = \mathrm{Obj}_{\mathcal{BVV}}$. □

THEOREM 4.3. *The optimal objective function value of the* $\mathcal{BVV}_{\mathrm{Imp2}}$ *formulation* (3.5)*,* (3.6) *and* (4.1) *is equal to the optimal objective function value of the* $\mathcal{BVV}_{\mathrm{Imp1}}$ *formulation* (3.5)*,* (3.6) *and* (3.8)*.*

PROOF. Let a feasible solution to the $\mathcal{BVV}_{\mathrm{Imp2}}$ formulation be represented by a vector $(\bar{x}'', \bar{y}'')$, $\bar{x}'' = (x_1'', \ldots, x_n'')$, $\bar{y}'' = (y_1'', \ldots, y_n'')$, $n = |V|$. Again, let a vector $(\bar{x}', \bar{y}')$ of variables x_i' ($\bar{x}' = (x_1', \ldots, x_n')$) and y_i' ($\bar{y}' = (y_1', \ldots, y_n')$) be defined such that $y_i' = y_i''$ for each $i \in V$ and

$$x_i' = \begin{cases} 0, & x_i'' \in [0,1) \\ 1, & x_i'' \in [1,+\infty) \end{cases}.$$

The variables x_i' and y_i' have binary values by this definition, therefore they satisfy conditions (3.8).

Assuming that $x_i'' + y_i'' + \sum_{j\in N_i} y_j'' \geqslant 1$, two cases arise:

1) $(y_i'' = 1) \vee (\exists j \in N_i)(y_j'' = 1)$,
2) $(y_i'' = 0) \wedge (\forall j \in N_i)(y_j'' = 0)$.

The first case implies that $y_i'' = 1$ or there exists j such that $y_j'' = 1$, i.e. $y_i'' + \sum_{j\in N_i} y_j'' \geqslant 1$. Now, knowing that $y_i' = y_i''$ and $x_i' \geqslant 0$ for each $i \in V$, it follows that $x_i' + y_i' + \sum_{j\in N_i} y_j' \geqslant 1$. The second case implies that $1 \leqslant x_i'' + y_i'' + \sum_{j\in N_i} y_j'' = x_i''$. Now, because $x_i'' \geqslant 1$ it follows that $x_i' = 1$. Further, because $y_i \in \{0,1\}$ it follows that $x_i' + y_i' + \sum_{j\in N_i} y_j' \geqslant 1$ meaning that conditions (3.2) are satisfied also.

Now, the objective function value of the $\mathcal{BVV}_{\mathrm{Imp1}}$ formulation can be calculated as $\sum_{i\in V} x_i' + 2\sum_{i\in V} y_i'$. Because of the relations $y_i' = y_i''$ and $x_i' \leqslant x_i''$ it is easy to notice that $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} \leqslant \mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp2}}}$.

Again, knowing that the objective function value of every relaxed minimization problem is less than or equal to the objective function value of the associated original problem, it follows that $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp2}}} \leqslant \mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}}$. Combining the given inequalities the theorem is proven, i.e. $\mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp1}}} = \mathrm{Obj}_{\mathcal{BVV}_{\mathrm{Imp2}}}$. □

Concerning $\mathcal{BVV}_{\mathrm{Imp1}}$ and $\mathcal{BVV}_{\mathrm{Imp2}}$ formulations, the number of constraints is reduced to $|V|$. Further more, by using $\mathcal{BVV}_{\mathrm{Imp2}}$ formulation $2|V|$ binary variables are substituted by $|V|$ binary and $|V|$ real variables.

5. Conclusions

This paper is devoted to the Roman domination problem. For the $\mathcal{RR}$ mathematical formulation it was proven that from a total of $2|V|$ variables, $|V|$ variables can be relaxed to be real. For the $\mathcal{BVV}$ mathematical formulation it was proven that a set of $|V|$ constraints can be excluded. Further more, the number of $2|V|$ binary variables of $\mathcal{BVV}$ mathematical formulation can be also relaxed to $|V|$ binary and $|V|$ real variables. While Theorems 4.1–4.3 shows that improved formulations $\mathcal{RR}_{Imp}$, $\mathcal{BVV}_{\text{Imp1}}$ and $\mathcal{BVV}_{\text{Imp2}}$ are equivalent to the existing ones, there are significant improvements of the computational efforts for solving these formulations.

Operating with lesser number of constraints and integer variables should provide a memory savings in solving the Roman domination problem on large size instances. For instance, on Intel® Core$^{\text{TM}}$ i7-4700MQ CPU @ 2.40GHz 2.39GHz with 8GB RAM under Windows 8.1 operating system and based on the new formulations, standard optimization solver CPLEX was able to find the optimal solution value on classes of graphs such as grid, net and planar up to 600 vertices.

Designing an exact method using the proposed MILP formulations is a matter of the future research. Also, in the future work it will be interesting to consider some variants of the Roman domination problem.

References

1. A. Bouchou, M. Blidia, *Criticality indices of roman domination of paths and cycles*, Australas. J. Comb. **56** (2013), 103–112.
2. A. P. Burger, A. P. de Villiers, J. H. van Vuuren, *A binary programming approach towards achieving effective graph protection*, Proc. 2013 ORSSA Annual Conf., ORSSA, 2013, pp. 19–30.
3. V. Currò, *The Roman Domination Problem on Grid Graphs*, Ph.D. thesis, Università di Catania, 2014.
4. P. A. Dreyer, Jr., *Applications and Variations of Domination in Graphs*, Tech. report, 2000.
5. A. Klobučar, I. Puljić, *Some results for roman domination number on cardinal product of paths and cycles*, Kragujevac J. Math. **38**(1) (2014), 83–94.
6. C. H. Liu, G. J. Chang, *Roman domination on strongly chordal graphs*, J. Comb. Optim. **26**(3) (2013), 608–619.
7. P. Pavlič, J. Žerovnik, *Roman domination number of the cartesian products of paths and cycles*, Electron. J. Comb. **19**(3) (2012), p. 19.
8. C. S. ReVelle, K. E. Rosing, *Defendens imperium romanum: a classical problem in military strategy*, Am. Math. Mon. **107**(7) (2000), 585–594.
9. W. Shang, X. Hu, *The roman domination problem in unit disk graphs*; in: *Computational Science – Iccs 2007: 7th Internat. Conf., Beijing China, May 27–30, 2007, Proc., Part III*, Lect. Notes Comput. Sci. 4489, Springer, 2007, pp. 305–312.
10. I. Stewart, *Defend the roman empire!*, Scientific American **281** (1999), 136–138.

Faculty of Mathematics
Department for Applied Mathematics
University of Belgrade
Belgrade
Serbia
marijai@math.rs

(Received 25 03 2015)
(Revised 14 10 2015)

Yugoslav Journal of Operations Research
28 (2018), Number 3, 291–314
DOI: https://doi.org/10.2298/YJOR161015018I

A NEW VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING DYNAMIC MEMORY ALLOCATION PROBLEM

Marija IVANOVIĆ
Faculty of Mathematics, University of Belgrade, Serbia
marijai@math.rs

Aleksandar SAVIĆ
Faculty of Mathematics, University of Belgrade, Serbia
asavic@matf.bg.ac.rs

Dragan UROŠEVIĆ
Mathematical Institute, SANU, Belgrade, Serbia
draganu@mi.sanu.ac.rs

Djordje DUGOŠIJA
Faculty of Mathematics, University of Belgrade, Serbia
dugosija@matf.bg.ac.rs

Received: October 2016 / Accepted: May 2018

Abstract: This paper is devoted to the Dynamic Memory Allocation Problem (DMAP) in embedded systems. The existing Integer Linear Programing (ILP) formulation for DMAP is improved, and given that there are several metaheuristic approaches for solving the DMAP, a new metaheuristic approach is proposed and compared with the former ones. Computational results show that our new heuristic approach outperforms the best algorithm found in the literature regarding quality and running times.

Keywords: Dynamic Memory Allocation Problem, Combinatorial Optimization, Metaheuristics, Variable Neighborhood Search.

MSC: 90C59, 05C90, 68T20.

1. INTRODUCTION

Dedicated computational systems within a larger mechanical or electrical system, usually with real time constraints, or the embedded systems, represent an integral part of a large number of devices, ranging from portable devices up to large industrial, medical, and military structures. Minimizing energy consumption while increasing reliability and performance present a major challenge for engineers. Therefore, designers want to find a balance between the architecture cost and its power consumption [1].

Power consumption of a given application can be reduced by using data access parallelization, which leads us to the definition of the Dynamic Memory Allocation Problem (DMAP).

DMAP can be described as follows. If an application involves a number of data structures with the given size and access cost, a number of memory banks with limited memory capacity, and one external memory bank with unlimited capacity, develop the best memory allocation scheme so that change of data structure allocation during program execution is allowed, while keeping loading, moving, and access time for those and all other operations to a minimum.

Given that modern devices are usually designed to run on processors with integrated or external memory, this problem has become very popular.

DMAP, as defined above, has an important role in modern software since the dynamic memory usage provides greater flexibility and functionality of the applications. There are a large number of papers and references to this particular issue in the literature. In this paper, DMAP in embedded systems is studied as an execution time problem, shown by Wuytack et al. in [17].

A memory allocation problem in embedded systems was widely analyzed by Soto et. al. in [14]. In [12] Soto et al. proposed the first Mixed Integer Linear Programing (MILP) formulation for the static version of the memory allocation problem and a metaheuristic approach based on the Variable Neighborhood Search (VNS). Later, Soto et al. [13] dealt with a dynamic version of the memory allocation problem, providing ILP formulation, and two iterative approaches for solving DMAP in embedded systems. These two approaches were followed by GRASP with the ejection chains for the memory allocation problem in embedded systems proposed in Sevaux et al. [11]. Focused on improving the memory allocation problem in embedded systems, Sánchez-Oro et. al [15] and [16] recently proposed a parallel VNS algorithm for DMAP and compared it with the previous ones. More precisely, they focused on the Synchronous Parallel VNS (SPVNS) variant that was used for parallelization of the local search method in the sequential VNS (in [15]) and on the Replicated Shaking VNS (RSVNS), which allows the search to simultaneously explore more solutions in the current neighborhood (in [16]).

In this paper, an improved ILP formulation for DMAP in embedded systems is proposed and a new metaheuristic approach based on the VNS is given. Results of this new metaheuristic approach are compared with the results from the existing literature.

The paper is organized as follows: Problem notation and ILP formulation known from the literature are given in Section 2. Section 3 is dedicated to the improved ILP formulation for dynamic versions of memory allocation problem followed by a short illustrative example. A new VNS approach is presented in the next section. Finally, the existing results from the literature, and the results achieved by using new VNS approach are given and compared in Section 5.

2. EXISTING MATHEMATICAL FORMULATION AND PROBLEM NOTATION

Notations and problem presentation will be similar as in [13]. Let us assume that the number of memory banks is limited by an electronic device, and that there are m internal memory banks with limited capacity, denoted by c_j, $0 < c_j < \infty$ ($j = 1, \ldots, m$), and one external memory bank whose size is supposed to be large enough so that it can be considered as unlimited ($c_0 = \infty$), i.e. the chosen memory architecture is similar to the one of TI C6201 device. Size of a memory bank is given in kilo Bytes (kB), and each memory bank is denoted by β_j, $j = 0, \ldots, m$. Without loss of generality, let us assume that a particular application is written in C++ code as a set of binary operations between the involved data structures. The term *data structure* is used for scalars, arrays, or similar structures of the applications. Further, let us assume that a set of all data structures is finite and depends on a program code. Furthermore, let us mark each data structure with α_i, where $i = 1, ..., n$, where n is a number of data structures. Data structure α_i size is defined in kB and marked by a_i. Due to the fact that external memory is a huge mass storage, all operations with data structures will be p times slower if mapped to the external memory bank. Therefore, we say that p is a penalty cost for loading data structure from the external memory bank. As all operations are not used at the same time, program execution time T is divided into t time blocks, I_t; it is not necessary that time blocks have constant size, and it is allowed to reconsider and move data structures before and after each time block. For instance, we say that a time block has size equal to one if it contains only one operation. If all time blocks are of size one, moving data structures is allowed before and after each operation. Since the access to data structures during the particular time block can differentiate, we define its access cost in milliseconds per kilo Bytes (ms/kB) and mark access cost to data structure α_i at time block I_t by $e_{i,t}$.

As previously mentioned, given that the parallel data structure usage can speed up program execution, moving data structure will be useful, especially because memory banks are of limited size. We assume that moving data structure is allowed only in between two time blocks and not during, without loss of generality. The problem with only one time block is a static memory allocation problem and more about it can be found in [8] and [12]. Moving data structures and rearranging their allocation is useful because in some cases moving data structure from external to internal memory bank and loading it from the internal memory bank can be less expensive than loading and operating with it while it is mapped to the external memory bank. Every data structure movement is also calculated.

It is assumed that the data structure movement between two internal memory banks costs l times its size, while the movement between internal and external memory banks is v times its size.

Similarly, when data structures are jointly involved in the same operation, i.e. have to be accessed at the same time, it will be assumed that data structure location affects its loading time. If two data structures, say α_s and α_d, are jointly involved in the same operation during the particular time block I_t, the given pair is called "conflict pair" with notation $\{\alpha_s, \alpha_d\}_t$. The number of conflict pairs at time block I_t, $t = 1, \ldots, T$ is denoted by k_t, while the set of all conflict pairs accessed at time block I_t is given by O_t. Accessing the conflict pair $\{\alpha_s, \alpha_d\}_t$ at time block I_t will cost $b_{\alpha_s,\alpha_d,t}$ if both data structures are mapped onto two different internal memory banks, $2b_{\alpha_s,\alpha_d,t}$ if both data structures are mapped onto the same internal memory bank, respectively; $pb_{\alpha_s,\alpha_d,t}$ if one data structure is mapped onto the external memory bank, and $2pb_{\alpha_s,\alpha_d,t}$ if both data structures are mapped onto the external memory bank.

Let us assume that set O_t consists of k_t conflict pairs, marked as $\{\alpha_{s_1}, \alpha_{d_1}\}$, $\{\alpha_{s_2}, \alpha_{d_2}\}, \ldots, \{\alpha_{s_{k_t}}, \alpha_{d_{k_t}}\}$. In order to reduce the number of indices, for r-th conflict pair $\{\alpha_{s_r}, \alpha_{d_r}\}$, $r = 1, \ldots, k_t$ instead of using notation $b_{\alpha_{s_r},\alpha_{d_r},t}$ conflict cost will be written as $b_{r,t}$. The set of all required data structures at time block t is denoted by P_t.

Again, it is assumed that all data structures are mapped to the external memory bank at the beginning of the program execution. Also, at the beginning of the program execution, the number of time intervals (T) is given and for each time interval I_t, $t = 1, \ldots, T$, sets P_t and O_t are followed with all necessary data (access cost for each data structure $\alpha_i \in P_t$ and conflict costs for each conflict pair $\{\alpha_{s_r}, \alpha_{d_r}\}, r = 1, \ldots, k_t$ in a particular time interval I_t).

The first mathematical formulation for DMAP in embedded systems is presented as it was proposed in [13].

For all $(i, j, k) \in \{1, \ldots, n\} \times \{0, \ldots, m\} \times \{1, \ldots, T\}$ a decision variable $x_{i,j,t}$ is set to one if and only if data structure α_i is allocated to the memory bank β_j during the time block I_t, $x_{i,j,t} = 0$ otherwise. Given that, the conflict pair is considered as closed if the involving data structures are mapped onto two different memory banks but open otherwise. For all conflicts $r \in \{1, \ldots, k_t\}$ at time block $I_t \in \{1, \ldots, T\}$, decision variable $y_{r,t}$ will be set to one if and only if during the time interval I_t conflict r is closed, otherwise $y_{r,t} = 0$. Further, data structure moving is represented by the two following sets of variables: for all $i \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, T\}$, $w_{i,t}$ is set to one if and only if data structure α_i has been moved from a memory bank $\beta_j \neq \beta_0$ to a different memory bank $\beta_{j'} \neq \beta_0$ between time blocks I_{t-1} and I_t. For all α_i, $i = \{1, \ldots, n\}$, at time block $I_t = \{1, \ldots, T\}$, $w'_{i,t}$ would be set to one if and only if data structure α_i was moved from internal memory bank to the external memory bank, or if it was moved from the external memory bank to an internal memory bank between time blocks I_{t-1} and I_t.

Before presenting ILP formulation for DMAP, let us recall all input and output data described in [13].

Input:

T - number of time intervals.

n - number of data structures.

m - number of internal memory banks.

p - penalty cost for operating with data structure when it is mapped to the external memory bank.

v - penalty cost for data structure movement from external memory bank to the internal and vice versa.

l - penalty cost for data structure movement between two different internal memory banks.

a_i - size of data structure α_i, $i = 1, ..., n$.

c_j - size of internal memory bank β_j, $j = 1, ..., m$.

P_t - set of data structures which are going to be used at the time block I_t, $t = 1, ..., T$.

$e_{i,t}$ - number of times that $\alpha_i \in P_t$, $i = 1, ..., |P_t|$ was accessed during the time block I_t, $t = 1, ..., T$.

O_t - set of conflict pairs for time block I_t, $|O_t| = k_t$, $t = 1, ..., T$.

$b_{r,t}$ - conflict cost for conflict pair $\{\alpha_{s_r}, \alpha_{d_r}\} \in O_t$, $r = 1, ..., k_t$ for time block I_t, $t = 1, ..., T$.

Output:

$x_{i,j,t}$ - the decision if data structure α_i $(i = 1, ..., n)$ is to be mapped to the memory bank β_j $(j = 0, ..., m)$ during the time interval I_t, $(t = 1, ..., T)$.

$y_{r,t}$ - the decision if conflict r is to be closed during time interval I_t $(t = 1, \ldots, T)$.

$w_{i,t}$ - the decision if data structure α_i, $i = 1, ..., n$ is to be moved from one internal memory bank to a different internal memory bank between time intervals I_{t-1} and I_t.

$w'_{i,t}$ - the decision if data structure α_i $(i = 1, ..., n)$ is to be moved from the external to the internal memory bank or vice versa between time intervals I_{t-1} and I_t.

Now, DMAP formulation known from the literature can be described as followed:

$$\min \quad f = \sum_{t=1}^{T}[(p-1)\sum_{\alpha_i \in P_t}(e_{i,t} \cdot x_{i,0,t}) - \sum_{r=1}^{k_t}(y_{r,t} \cdot b_{r,t}) + \sum_{\alpha_i \in P_t} a_i(l \cdot w_{i,t} + v \cdot w'_{i,t})] \quad (1)$$

Subject to the constraints

$$\sum_{j=0}^{m} x_{i,j,t} = 1, \qquad i \in \{1,...n\}, \quad t \in \{1,...,T\} \quad (2)$$

$$\sum_{\alpha_i \in P_t} x_{i,j,t} \cdot a_i \le c_j, \qquad j \in \{1,...,n\}, t \in \{1,...,T\} \quad (3)$$

$$x_{s_r,j,t} + x_{d_r,j,t} \le 2 - y_{r,t} \qquad (\alpha_{s_r}, \alpha_{d_r}) \in P_t, j \in \{0,...,n\}, r \in \{1,...,k_t\}, t \in \{1,...,T\} \quad (4)$$

$$x_{i,j,t-1} + x_{i,j',t} \le 1 + w_{i,t} \qquad i \in \{1,\dots,n\}, (j,j') \in \{1,...,m\}^2, (\beta'_j \ne \beta_j), t \in \{1,...,T\} \quad (5)$$

$$x_{i,0,t-1} + x_{i,j,t} \le 1 + w'_{i,t} \qquad i \in \{1,...,n\}, j \in \{1,...,m\}, t \in \{1,...,T\} \quad (6)$$

$$x_{i,j,t-1} + x_{i,0,t} \le 1 + w'_{i,t} \qquad i \in \{1,...,n\}, j \in \{1,...,m\}, t \in \{1,...,T\} \quad (7)$$

$$x_{i,j,0} = 0 \qquad i \in \{1,...,n\}, j \in \{1,...,m\} \quad (8)$$

$$x_{i,0,0} = 1 \qquad i \in \{1,...,n\} \quad (9)$$

$$x_{i,j,t} \in \{0,1\} \qquad i \in \{1,...,n\}, j \in \{1,...,m\}, t \in \{1,...,T\} \quad (10)$$

$$w_{i,t} \in \{0,1\} \qquad i \in \{1,...,n\}, t \in \{1,...,T\} \quad (11)$$

$$w'_{i,t} \in \{0,1\} \qquad i \in \{1,...,n\}, t \in \{1,...,T\} \quad (12)$$

$$y_{r,t} \in \{0,1\} \qquad r \in \{1,...,k_t\}, t \in \{1,...,T\} \quad (13)$$

Constraints (2) state that every data structure is allocated to only one memory bank at time interval I_t. Constraints (3) ensure that the total size of all data structures allocated to any memory bank at time interval I_t does not exceed its size. Constraints (4) - (7) ensure that variables y_{rt}, $w_{i,t}$ and $w'_{i,t}$ are set appropriately. The initial conditions are given by the constraints (8) and (9) and finally, constraints (10)-(13) enforce binary requirements.

Note that the presented mathematical formulation does not correspond to the DMAP problem. More precisely, calculation of the cost function is based only on the fact that data structures are mapped to the same memory bank, though, both data structures can be mapped to the external memory bank or to the same internal memory bank. Additionally, the cost of accessing data structure α_i during time interval I_t is calculated as $(p-1)e_{i,t}$ if the data are mapped to the external memory bank and as 0 if the data are mapped to the internal memory bank, which is not correct. We believe that $\sum_{t=1}^{T}\sum_{\alpha_i \in P_t} e_{it}$ is unintentionally left out, given the fact that with this constant, the cost function (1) indeed corresponds to the DMAP problem. Still, the way the solution to the DMAP problem was calculated in [11], [13] and [15] included before mentioned oversights.

Considering that it was not easy to notice that the mentioned constant was left out, we have given an improved ILP formulation for the DMAP in embedded system in the following section.

3. IMPROVEMENT TO THE EXISTING ILP FORMULATION FOR DMAP IN EMBEDDED SYSTEMS

Let binary variables $x_{i,j,t}$, $w_{i,t}$ and $w'_{i,t}$ $(i = 1, \ldots, n,\ j = 0, \ldots, m,\ t = 1, \ldots, T)$ have the same meaning as before, and let binary variables $y_{r,t}$, defined such that r represent a conflict pair $(\alpha_{s_r}, \alpha_{d_r})$ at time block I_t, have slightly different meaning, i.e.

$$y_{r,t} = \begin{cases} 1, & \text{if data } \alpha_{s_r} \text{ and } \alpha_{d_r} \text{ are both mapped into} \\ & \text{the same memory bank at time block } I_t \\ 0, & \text{otherwise} \end{cases} \quad (\alpha_{s_r}, \alpha_{d_r}) \in I_t,\ t = 1, \ldots, T \quad (14)$$

Additionally, let e_{it} represent the cost for accessing data α_i during the time block I_t instead of denoting the number of that data structure α_i accessed during the time block I_t. Value of the e_{it} will actually differ, concerning the previous definition by the conflict cost where it is involved during time interval I_t. We think that with this definition, loading each data structure and accessing the same data structure during the conflict, in which it is involved, are defined more precisely.

Now, the improved formulation for DMAP can be defined as follows:

$$\min \quad f = \sum_{t=1}^{T} \Bigg(\sum_{\alpha_i \in P_t} \Big(1 + (p-1)x_{i,0,t}\Big)e_{i,t} +$$

$$\sum_{\substack{r=1 \\ \{\alpha_{s_r}, \alpha_{d_r}\} \in O_t}}^{k_t} b_{r,t}\Big(1 + y_{r,t} + (p-1)\Big(x_{s_r,0,t} + x_{d_r,0,t}\Big)\Big) + \sum_{i=1}^{n} a_i\Big(lw_{i,t} + vw'_{i,t}\Big)\Bigg) \quad (15)$$

subject to constraints:

$$x_{s_r,j,t} + x_{d_r,j,t} \leq y_{r,t} + 1 \qquad (\alpha_{s_r}, \alpha_{d_r}) \in P_t, \quad t = 1, \ldots, T \quad (16)$$

$$y_{r,t} \in \{0, 1\} \quad (17)$$

and constraints (2), (3), (5)-(12) from the existing ILP formulation.

Constraints (16) correspond to the constraints (4) in accordance with the new definition of variables $y_{r,t}$.

Access cost at particular time block I_t is equal to $\sum_{\alpha_i \in P_t}(1 + (p-1)x_{i,0,t})e_{i,t}$, which covers cases when data structure is mapped to the external memory bank. At the same time block, expression $1 + y_{r,t} + (p-1)(x_{s_r,0,t} + x_{d_r,0,t})$ is equal to 1 if data structures from the same conflict pair are mapped onto two different internal memory banks; it is equal to 2 if they are mapped onto the same internal memory bank, respectively, and it is equal to p if one conflict data is mapped to the external memory bank and is equal to $2p$ if both data structures are mapped to the external memory bank. Therefore,

$$\sum_{\substack{r=1 \\ \{\alpha_{s_r},\alpha_{d_r}\}\in O_t}}^{k_t} b_{r,t}\Big(1 + y_{r,t} + (p-1)(x_{s_r,0,t} + x_{d_r,0,t})\Big)$$

correspond to conflict cost at time interval I_t, while movement cost between two time blocks is again calculated by the sum

$$\sum_{t=1}^{T}\sum_{i=1}^{n} a_i\Big(lw_{i,t} + vw'_{i,t}\Big)$$

Now, given that all oversights are covered by the new proposed cost function, the improved ILP formulation corresponds to the proposed DMAP in embedded systems.

For $T = 1$, DMAP formulation becomes ILP formulation, which was proposed and proven to be feasible for static version of the memory allocation problem in [8]. Let us illustrate DMAP and its ILP formulation on a short example.

An illustrative example is given on a small size instance.

Example 3.1. *There are 9 data structures (α_i, $i = 1,..,.,9$), which have to be placed into 2 internal memory banks (β_1 and β_2), each size of 1000 (kB) ($c_j = 1000$, $j = 1,2$), and 1 external memory bank ($c_0 = \infty$). Sizes of data structures are given in Table 1. Loading time is divided into three time blocks of specified sizes. Data structures are used in pairs. Access cost for each data structure at each time block, together with the conflict sets are given in Table 2. For instance, at time block I_1, 4 data structures are used, $P_1 = \{\alpha_2, \alpha_5, \alpha_8, \alpha_9\}$, with zero access time each and with two conflict pairs. The first one is between data structures α_8 and α_9 ($k_{1,1} = (\alpha_8, \alpha_9)$) with conflict cost $b_{1,1} = 592$, and the second one is between data structures α_2 and α_5 ($k_{2,1} = (\alpha_2, \alpha_5)$) with conflict cost $b_{2,1} = 192$. Loading all operations from the external memory bank costs $p = 16$ times more than loading them from the internal memory bank. Moving data between two internal memory banks costs $l = 1$ (ms/kB), and moving between internal and external memory bank $v = 4$ (ms/kB).*

Let us assume that at the beginning of the program ($t = 0$), all data structures are mapped to the external memory bank. Knowing the order of the data structure usage, we can move them before each time block in order to reach better results. Table 3 represents the optimal solution obtained by using IBM ILOG CPLEX optimization solver for a mathematical model presented above. Solution can be interpreted as follows: at time block I_1, data structures $\alpha_1, \alpha_3, \alpha_4, \alpha_6$, and α_7 should be mapped to the external memory bank, marked as β_0, while data structures α_5 and α_8 should be mapped to the memory bank marked as β_1, and finally, data structures α_2 and α_9 should be mapped to the memory bank marked as β_2. Then, total execution time, which consists of moving time and conflict time for each

Table 1: Data structure size

data structures	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9
size of data sruct.	256	4	496	256	4	256	256	256	256

Table 2: Data access and conflict costs for $t = 1$ (left) , $t = 2$ (middle) and $t = 3$ (right)

	Time block: 1				Time block: 2				Time block: 3		
Data set	α_2	α_5	α_8	α_9	α_1	α_2	α_3	α_4	α_4	α_6	α_7
Access c.	4	4	256	256	256	4	496	256	256	256	256
	Conflicts				Conflicts				Conflicts		
Conflicts	α_s	α_d	Cost		α_s	α_d	Cost		α_s	α_d	Cost
$k_{1,t}$	α_8	α_9	592		α_2	α_2	576		α_6	α_7	1,024
$k_{2,t}$	α_2	α_5	192		α_1	α_4	64		α_4	α_7	1,024
$k_{3,t}$					α_1	α_3	1,024		α_4	α_4	64

time block, is presented in Table 4. Loading time is excluded from this illustration because loading cost is equal to zero for each data structure at each time block. For instance, total block costs will be explained in detail for the time block I_3: given that data structure α_1 is replaced with data structure α_6 and moved from memory allocation β_2 to the memory allocation β_0 and the other way around, moving cost is equal to $(256 + 256) \cdot 4 = 2048$ (ms). Further, conflict costs for the same block is equal to $1,024 + 1,024 + 64 \cdot 2 = 2,176$ (ms), which brings total block cost to $4,224$ (ms). Combining the results for all three time blocks gives the problem execution cost of $17,772$ (ms).

Table 3: Solution to the example for DMAP formulation

Mem. allocation	t = 0	t = 1	t = 2	t = 3
β_0	All data	$\alpha_1, \alpha_3, \alpha_4, \alpha_6, \alpha_7$	α_6, α_8	α_1, α_8
β_1		α_5, α_8	$\alpha_3, \alpha_5, \alpha_7$	$\alpha_3, \alpha_5, \alpha_7$
β_2		α_2, α_9	$\alpha_1, \alpha_2, \alpha_4, \alpha_9$	$\alpha_2, \alpha_4, \alpha_6, \alpha_9$

Table 4: Costs

	t = 1	t = 2	t = 3
Access cost	520	1,012	768
Moving cost	2,080	6,080	2,048
Conflict cost	784	2,304	2,176
Total cost per block	3,384	9,396	4,992

Now, the solution in terms of the proposed ILP formulation can be interpreted as follows: $\min f = 17,772$ whith nonzero decision variables $x_{1,0,1}$, $x_{3,0,1}$, $x_{4,0,1}$, $x_{6,0,1}$, $x_{7,0,1}$, $x_{5,1,1}$, $x_{8,1,1}$, $x_{2,2,1}$, $x_{9,2,1}$, $x_{6,0,2}$, $x_{8,0,2}$, $x_{3,1,2}$, $x_{5,1,2}$, $x_{7,1,2}$, $x_{1,2,2}$, $x_{2,2,2}$, $x_{4,2,2}$, $x_{9,2,2}$, $x_{1,0,3}$, $x_{8,0,3}$, $x_{3,1,3}$, $x_{5,2,3}$, $x_{7,2,3}$, $x_{2,3,3}$, $x_{4,2,3}$, $x_{6,2,3}$, $x_{9,2,3}$, $y_{1,2}$, $y_{2,2}$, $y_{3,3}$, $w'_{2,1}$, $w'_{5,1}$, $w'_{8,1}$, $w'_{9,1}$, $w'_{1,2}$, $w'_{3,2}$, $w'_{4,2}$, $w'_{7,2}$, $w'_{8,2}$, $w'_{1,3}$, and $w'_{6,3}$.

In order to compare the presented ILP formulation for the DMAP with the one known from the literature, both formulations were coded in C++ and tested on Intel(R) Core(TM) i7-4700Mq CPU @ 2.40GHz, 2394MHz, 4 Core(s) with 8GB

RAM with CPLEX 12.6. For experimental testings of the proposed ILP formulations we used the same set of instances as the one reported by Soto et. al in [13]. The set of instances can be downloaded under the name
dmap.zip from http://www.optsicom.es/dmap/dmap.zip.
More details about these instances are given in Section 5.

In these testings, penalty cost p is set to be equal to 16ms/kB, movement data structure cost from internal to external memory bank (v) and vice versa is set to be equal to 4ms/kB, while the movement between two internal memory banks (l) is set to be equal to 1ms/kB. Comparison of the presented Soto et al. [13] formulation and our formulation is presented in Table 5. The name of an instance is given in the first and the fifth column. Solution value obtained for the particular instance by using ILP formulation presented in [13] is given in the second and the sixth column. Solution values for the corrected cost function of the same ILP formulation, as described at the end of the Section 2, are given in the third and the seventh column. Finally, solution values of the tested instances by using our formulation are given in the fourth and the eight column. Time limit for all testings was set to 7200 sec. Sign "*" was used in case that testings were stopped earlier because of status "out of memory". On majority of instances, the optimal solution value was found using both formulations. On instances such as "myciel4dy.col"-"r125.1dy.col", the found solutions were not proved to be the optimal. As it can be seen from Table 5, on instances where optimal solution was not reached, CPLEX was able to find better results by using our ILP formulation on 4 instances and on 1 by using Soto et al. [13] corrected formulation.

4. VNS FOR THE PROPOSED DMAP PROBLEM

The problem of combinatorial optimization, presented in this paper, can be solved by using exact methods, but because of its number of variable limitations for large scale variables, metaheuristics are more useful. Therefore, DMAP was solved by using VNS hybridized with Tabu Search (TS) in [13] and GRASP with ejection chains in [11]. In this paper, a VNS metaheuristic method hybridized with Variable Neighborhood Descent (VND) search, which corresponds to the proposed DMAP model, is presented. VNS metaheuristic was first presented by Mladenović and Hansen in [9], and the main idea was proposed in 1995, [10]. Later, it was followed with several papers aimed at improving the method, [2, 3, 4, 5, 6], and [7]. The main idea of this heuristic is that starting from one initial solution by using systematic search, we move to the solution that is locally the best.

Solution space. Each solution can be represented by a matrix (two-dimensional array) Z with T rows and n columns. Value of element $z_{t,i}$ represents a label of a memory bank on which the data structure α_i is allocated at the time interval I_t ($t = 1, 2, ..., T$, $i = 1, 2, ..., n$). Only feasible solutions are considered, i.e. data structures are allocated in such a way that there are no overloaded banks. In other words, for each time interval I_t and each bank β_j, the following condition is

Table 5: Comparison of the ILP formulation for the DMAP presented in Soto et al [13], their formulation with correction and our ILP formulation

	from Soto et al [13]				from Soto et al [13]		
	without	with	our ILP		without	with	our ILP
	correct val.	correct val.	formulat.		correct val.	correct val	formulat.
Instance	*val*	*val*	*val*	Instance	*val*	*val*	*val*
adpcmdy	37368	44192	44192	mulsol_i2dy	*	*	*
alidydy	*	107518	107518	mulsol_i4dy	*	*	*
cjpegdy	-4237207	4466800	4466800	mulsol_i5dy	*	*	*
compressdy	-222272	342592	342592	myciel3dy	*	6379	6379
fpsol2i2dy	*	*	*	myciel4dy	*	**18225**	18295
fpsol2i3dy	*	*	*	myciel5dy	*	40380	40380
gsm_newdy	7320	7808	7808	myciel6dy	*	108726	108726
gsmdy	548645.75	1355390	1355390	myciel7dy	*	*	*
gsmdycorrdy	-312628	494119	494119	queen5_5dy	*	21859	21859
inithx_i1dy	0	*	*	queen6_6dy	*	43306	**39144**
lmsbdy	-7245829	7409669	7409669	queen7_7dy	*	88652	**85606**
lmsbv01dy	-4051636	4350640	4350640	queen8_8dy	*	*	**138939**
lmsbvdy	-4061214	4323294	4323294	r125.1cdy	*	*	*
lmsbvdyexpdy	-4035252	4367024	4367024	r125.1dy	*	62354	**60931**
lpcdy	23802	26888	26888	r125.5dy	*	*	*
mpeg2enc2dy	7788.8586	9812.312	9812.312	spectraldy	6352	15472	15472
mpegdy	5805.625	10613.63	10613.63	treillisdy	1331.563	1805.563	1805.563
mug100_1dy	12797	29847	29847	turbocodedy	847	3195	3195
mug100_25dy	11621	28429	28429	volterrady	166	178	178
mug88_1dy	10227	25305	25305	zeroin_i1dy	*	*	*
mug88_25dy	9157	24181	24181	zeroin_i2dy	*	*	*
mulsol_i1dy	*	*	*	zeroin_i3dy	*	*	*

satisfied:

$$\sum_{i:z_{t,i}=j} a_i \leq c_j.$$

Note that the above expression represents the total size of memory bank β_j occupied by data structures allocated to that bank during the time interval I_t.

Similarly, a solution can be represented by a matrix X, where $x_{i,j,t} \in X$ is equal to one if data α_i is allocated to the memory bank β_j at the time block I_t and zero otherwise. For a matrix Z, which represents the solution, it is possible to evaluate the solution X and its respective objective function value $f(X)$. The objective function value consists of two parts:

- Costs of moving data structures from one memory bank to another,

- Costs of accessing conflict pairs.

Complexity of computing the first part of the objective function is $O(nT)$, while the cost of computing the second part is $O(K)$, where K represents a total number of conflict pairs ($K = \sum_{t=1}^{T} k_t$).

In the solution space, we introduce two types of moves:

- Insertion move,

- Swap move.

Insertion move consists of picking one time interval I_t and one data structure α_i, and moving that data structure from the bank β_{j_1} on which it is allocated in the current solution to the bank β_{j_2} ($j_2 \neq j_1$). Understandably, it is necessary to choose β_{j_2} in such a way that the new solution obtained by this move is feasible. If a solution is represented by a matrix Z, this move consists of setting value of element $z_{t,i}$ to j_2.

Swap move consists of picking time interval I_t, two data structures α_{i_1} and α_{i_2} currently allocated to different banks and exchanging their allocations. If a solution is represented by a matrix Z, this move consists of exchanging values of elements z_{t,i_1} and z_{t,i_2}.

Hence, based on the introduced moves, we can define neighborhoods of the solution space. So, neighborhood $\mathcal{N}_k(X)$ consists of all solutions X' obtained by applying k successive moves (Insertion or Swap move) starting from the solution X. Also, we introduce two neighborhoods, $N_{ins}(X)$ and $N_{swap}(X)$, as the sets of all solutions obtained by applying Insertion move or Swap move (respectively) on solution X. Note that the union of neighborhoods N_{ins} and N_{swap} represents the neighborhood $\mathcal{N}_1$.

Now, VNS heuristic can be defined in such a way that starting from initial feasible solution X it "shakes" that solution by creating another feasible solution $X' \in \mathcal{N}_k(X)$ and applies local search method in order to move to a better solution X''. If such a solution is not better than the current incumbent X, we create another neighborhood set $\mathcal{N}_{k+1}(X)$ and seek for a better solution until k reaches its maximum k_{max}. If X'' is better than a current incumbent solution, it becomes new incumbent and k becomes k_{min}. This systematic local environment change is needed because of the fact that local minimum within one environment does not have to be local minimum of the other environment, but a global minimum is a local minimum relative to all environments. Changing environments also enables to get out from the local minimum. VNS metaheuristic is illustrated with Algorithm 1.

Algorithm 1: Variable Neighborhood Search metaheuristics

```
1  $X \leftarrow InitialSolution()$;
2  $X_{opt} \leftarrow X$;
3  $f_{opt} \leftarrow f(X)$;
4  repeat
5      $k \leftarrow k_{min}$;
6      repeat
7          $X' \leftarrow Shake(X,k)$;
8          $X'' \leftarrow LocalSearch(X')$;
9          if $f(X'') < f_{opt}$ then
10             $X \leftarrow X''$;
11             $f_{opt} \leftarrow f(X)$;
12             $k \leftarrow k_{min}$;
13         else
14             $k \leftarrow k + k_{step}$;
15         end
16     until $k > k_{max}$;
17 until StoppingCondition();
```

Initial solution. At the beginning, all data structures are mapped to the external memory bank. After that, for each data structure α_i and each time interval I_t, we try to "reallocate" α_i to one of the banks such that the obtained solution is feasible and better than the incumbent. The pseudocode for calculation of an initial solution is given in Algorithm 2.

Shaking. Shaking in Neighborhood $\mathcal{N}_k$ ($Shake(X,k)$) consists of performing k randomly selected moves. For each of these k moves, Insertion move or Swap move were chosen equally (with probability equal to $p = 0.5$). The pseudo code for Shaking procedure is given in Algorithm 3.

Algorithm 2:

```
Function $InitialSolution()$;
for $\leftarrow 1$ to $n$ do
    for $t \leftarrow 1$ to $T$ do
        $X_{t,i} \leftarrow 0$;
    end
end
$fc \leftarrow f(X)$;
for $i \leftarrow 1$ to $n$ do
    for $t \leftarrow 1$ to $T$ do
        for $j \leftarrow 1$ to $m$ do
            $xo \leftarrow X_{t,t}$;
            $X_{t,i} \leftarrow j$;
            if $Feasible(X)$ and $f(X) < fc$ then
                $fc \leftarrow f(X)$;
            else
                $X_{t,i} \leftarrow xo$;
            end
        end
    end
end
return $X'$;
```

Algorithm 3:

```
Function $Shake(X, k)$;
$X' \leftarrow X$;
for $i \leftarrow 1$ to $k$ do
    $p \leftarrow RandomNumber(0, 1)$;
    if $p \leq 0.5$ then
        InsertMethod:
        repeat
            $t_r \leftarrow RandomNumber(1, T)$;
            $i_r \leftarrow RandomNumber(1, n)$;
            repeat
                $j_r \leftarrow RandomNumber(0, m)$;
            until $j_r \neq CurrentPosition(i_r, t_r)$;
        until $NotFeasibleSolution$;
        Move data structure $\alpha_{i_r}$ to the memory location $\beta_{j_r}$ at time block $I_{t_r}$.
    else
        SwapMethod:
        repeat
            $t_r \leftarrow RandomNumber(1, T)$;
            $i_{r_1} \leftarrow RandomNumber(1, n)$;
            repeat
                $i_{r_2} \neq i_{r_1}$
            until $i_{r_2} \neq i_{r_1}$;
        until $NotFeasibleSolution$;
        Swap memory locations for data structures $\alpha_{i_{r_1}}$ and $\alpha_{i_{r_2}}$ at time block
          $I_t$.
    end
end
```

Local Search is implemented as the Variable Neighborhood Descent (VND) method based on two previously defined moves (and corresponding neighborhoods N_{ins} and N_{swap}). This means that the neighborhood $N_{ins}(X')$ of a current solution X' is examined in order to find a solution X'_1, which is better than the solution X' ($f(X'_1) < f(X')$). If such a solution X'_1 exists, the corresponding move is performed and examining the neighborhood N_{ins} of the new solution continues. If there is no better solution in the neighborhood N_{ins} of the incumbent, examining the neighborhood N_{swap} of the incumbent continues. If there is a solution $X'_1 \in N_{swap}(X')$, the corresponding move is performed and examining the neighborhood N_{swap} of the new solution continues. If during examination of neighborhoods N_{swap} at least one improvement is made, we return to neighborhood N_{ins}, otherwise local search (VND) is finished. The pseudo-code of VND and included functions are given in Algorithms 4, 5, and 6.

Algorithm 4: Function for Variable Neighborhood Descent procedure

```
Function VND(X);
k ← 1;
X' ← X;
while k ≤ 2 do
    if k = 1 then
        X'' ← LSIns(X);
        if f(X'') < f(X') then
            X' ← X'';
            k ← 1;
        else
            k ← 2;
        end
    else
        X'' ← LSSwap(X);
        if f(X'') < f(X') then
            X' ← X'';
            k ← 1;
        else
            k ← 3;
        end
    end
end
return X';
```

Algorithm 5: Function for local search in neighborhood N_{ins}

```
Function LSIns(X);
X' ← X;
for i ← 1 to n do
    for t ← 1 to T do
        X'' ← Move(X', i, t);
        if Feasible(X'') and f(X'') < f(X') then
            X' ← X'';
        end
    end
end
return X';
```

Algorithm 6: Function for local search in neighborhood N_{swap}

```
Function LSSwap(X);
X' ← X;
for i_1 ← 1 to n do
    for i_2 ← i_1 + 1 to n do
        for t ← 1 to T do
            X'' ← Swap(X', i_1, i_2, t);
            if Feasible(X'') and f(X'') < f(X') then
                X' ← X'';
            end
        end
    end
end
return X';
```

Let us discuss briefly the complexity of the proposed local search. Cardinality of the neighborhood $N_{ins}(X')$ of the solution X' is nmT (each of n data structures can be reallocated in each of T time intervals to one of the remaining m memory banks). On the other hand, examination of all neighboring solutions X'_1 (suppose that X'_1 is obtained by moving data structures α_i from bank β_{j_1} to bank β_{j_2} during the time interval I_t) consists of calculating changes in objective function value.

The change of objective function value consists of

- change in the part of the objective function containing cost of transfering data structure from one bank to another (can be calculated in ($O(1)$)).
- change in the part of the objective function containing costs of accessing conflicts (can be calculated in $O(k_{t,i})$, where $k_{t,i}$ is number of conflicts involving data structure α_i in time interval I_t)

Obviously, the second part depends on a number of conflicts involving specified data structure and specified time interval. However, it is possible to make aggregate complexity analysis if we note that each conflict participates in change of the objective function for at most $2m$ neighboring solutions (from neighborhood N_{ins}). Finally, the total complexity for calculating the change of the second part of the objective function is $O(Km)$, while the total complexity for both parts is $O(nmT + Km)$.

Regarding neighborhood N_{swap}, cardinality of neighborhood is $\binom{n}{2}T$ (each of $\binom{n}{2}$ pairs of data structures can exchange allocation on each of the T time intervals). Similarly, calculating change in objective function contains two parts: change in cost of moving data structure (can be calculated in $O(1)$ time), and change of cost for conflict. Each conflict participates in change of the objective function for at most $2(n-2)+1$ moves (each of data structures from conflict pair can exchange allocation with any of the remaining data structures). Now, it can be concluded that the total complexity is $O\left(\binom{n}{2}T + Kn\right)$.

5. COMPUTATIONAL RESULTS

Experimental results obtained by the proposed VNS algorithm for solving DMAP are given in this section. VNS heuristic was coded in C++. All computational experiments were performed on Pentium Core Duo CPU @ 2.66GHz with 6GB RAM. For the experimental testings of the proposed implementation, the set of instances used is the same as the one reported by Soto in [13].
The set of instances can be downloaded under the name
dmap.zip from http://www.optsicom.es/dmap/dmap.zip.

Instances are classified by their name and three relevant characteristics, notably time interval, number of data structures, and number of internal memory banks. In the following table, the main features of the instances are shown: Instance name, number of time intervals (T), number of data structures (n), and number of available internal memory banks (m).

Instance name	T	n	m	Instace name	T	n	m
adpcmdy.dat	3	10	2	mulsol_i2dy.dat	39	188	16
alidydy.dat	48	192	6	mulsol_i4dy.dat	39	185	16
cjpegdy.dat	4	11	2	mulsol_i5dy.dat	40	185	16
compressdy.dat	3	6	2	myciel3dy.col	4	11	2
fpsol2i2dy.dat	87	451	15	myciel4dy.col	7	23	3
fpsol2i3dy.dat	87	425	15	myciel5dy.col	6	47	16
gsm_newdy.dat	2	6	2	myciel6dy.col	11	95	2
gsmdy.dat	5	19	2	myciel7dy.col	24	191	4
gsmdycorrdy.dat	5	19	2	queen5_5dy.col	5	25	3
inithx_i1dy.dat	187	864	27	queen6_6dy.col	10	36	4
lmsbdy.dat	3	8	2	queen7_7dy.col	16	49	4
lmsbv01dy.dat	4	8	2	queen8_8dy.col	24	64	5
lmsbvdy.dat	3	8	2	r125.1cdy.col	75	125	23
lmsbvdyexpdy.dat	4	8	2	r125.5dy.col	38	125	18
lpcdy.dat	4	15	2	r125.1dy.col	6	125	3
mpeg2enc2dy.dat	12	130	2	spectraldy.dat	3	9	2
mpegdy.dat	8	68	2	treillisdy.dat	6	33	2
mug100_1dy.col	7	100	2	turbocodedy.dat	4	12	3
mug100_25dy.col	7	100	2	volterrady.dat	2	8	2
mug88_1dy.col	6	88	2	zeroin_i1dy.dat	41	211	25
mug88_25dy.col	6	88	2	zeroin_i2dy.dat	35	211	15
mulsol_i1dy.dat	39	197	25	zeroin_i3dy.dat	35	206	15

Further, in order to compare with the known results, penalty cost p is set to be equal to 16ms/kB, movement data structure cost from internal to external memory bank (v), and vice versa, is set to be equal to 4ms/kB, while movement between two internal memory banks (l) is set to be equal to 1ms/kB.

Given that all metaheuritics, including VNS, have a stochastic nature, VNS algorithm was run 20 times for each problem instance. Finishing criteria of the proposed VNS were either time limit or event when k_{max} was reached. In this implementation, k_{max} was set to 10, k_{step} to 1, and k_{min} to 1. Execution time limit for VNS was set to $7,200$ seconds per instance.

For easier comparison, the results of the presented VNS metaheuristic on tested instances are summarized in Tables 6 - 9, where presented VNS algorithm is compared with IM (from [13]), CPA, GRASP, and GRASP+EC (from [11]) and with BVNS, and RVNS methods (from [16]).

Table 6 is organized as follows. Names of instances, which are sorted alphabetically, are given in the first column. Determined by the solution values presented in columns "from Soto et. al [13] with correct. val." and "our ILP formulat." of Table 5 and by the solution values obtained by all considered methods (IM, CPA, GRASP, GRASP+EC, BVNS and RSVNS, and VNS), the best known solution value is presented in the second column. The next six columns are obtained using results from Sevaux et. al [11], where all known methods for solving DMAP (IM, CPA, GRASP and GRASP+EC) till that time were compared. Results in corresponding columns "%dev" are presented as a deviation value from the best known value from the second column (in percentage). Respective running times, which are given only for GRASP and GRASP+EC methods, are copied from [11]. Solution values, respective running times (in seconds), and deviations from the best knowns (in percentage), obtained by VNS metaheuristic proposed in this paper, are presented in three final columns.

Table 6: Comparison between new and existing methods from the literature for solving DMAP

		IM	CPA	GRASP		GRASP+EC		VNS		
Instance	Best known	% dev.	% dev.	% dev.	time	% dev.	time	Obj.val.	time	% dev.
adpcmdy	44192	0	45.44	0	0.01	0	0	44192	0	0
alidydy	107398	939.7749	149.3545	159.5870	160.48	50.5519	85	107398	180.05	0
cjpegdy	4466792	0.0002	1.9302	0.0002	0.01	0.0002	0	4466800	0.01	0.0002
compressdy	342592	0	7.77	2.67	0.01	0	0	342592	0	0
fpsol2i2dy	2587875	64.4663	33.5580	7.9954	1015.13	7.9954	1000	2605980	835.11	0.6996
fpsol2i3dy	2582988	60.1959	33.7942	6.9327	1062.37	6.9327	1000	2616849	1000	1.3109
gsm_newdy	7808	0	17295	0	0.01	0	0	7808	0	0
gsmdy	1355389.875	<0.0001	<0.0001	0.1300	0.01	0.1300	0	1355390	0.22	<0.0001
gsmdycorrdy	494118	0	0	0.35	0.04	0.36	0	494118	1.85	0
inithx_i1dy	5652213	81.8834	24.2372	11.1145	700	11.3479	1000	5716414	1000	1.1359
lmsbdy	7409660	0.0001	0.3601	0.3301	0.29	0.1401	0	7409669	0	0.0001
lmsbv01dy	4350588	0.0012	3.7712	1.1312	0.01	1.8812	1000	4350640	0	0.0012
lmsbvdy	4323116	0.0041	2.2742	0.0041	0.01	1.1442	0	4323294	0.03	0.0041
lmsbvdyexpdy	4366972	0.0012	3.3812	2.6312	0.01	1.8812	1000	4367024	0.01	0.0012
lpcdy	26888	0	43.67	22.02	0.02	26.19	0	26888	0.04	0
mpeg2enc2dy	9812	0	45.99	9.46	0.75	10.14	0	10915.28	0.23	11.2442
mpegdy	10613.625	0.15	41.75	4.62	0.13	26.54	0	10648.88	480.44	0.3322
mug100_1dy	28890	0	109.98	25.49	14.71	21.47	0	30638	395	6.0505
mug100_25dy	28429	7.2813	101.1632	13.6967	11.89	14.7910	0	28876	323.07	1.5723
mug88_1dy	25305	0.877297	94.82432	10.18827	11.43	13.72906	0	25570	471.81	1.0472
mug88_25dy	24181	1.448331	74.51606	1.448331	7.78	0.533477	0	24365	392.18	0.7609
mulsol_i1dy	459598	177.7919	223.0343	73.8314	1096.13	12.7677	66	478739	1794.61	4.1647
mulsol_i2dy	552215	130.3367	214.9069	48.0897	1086.69	18.5287	71	557552	1792.89	0.9665
mulsol_i4dy	505927	132.3606	222.5533	37.8263	1057.35	12.7690	56	505927	1785.29	0
mulsol_i5dy	518854	144.7968	208.6101	40.8412	1080.78	10.7678	59	520929	1797.09	0.3999
myciel3dy	6379	89.14	6.88	8.42	1.24	11.26	0	6379	0.2	0
myciel4dy	18225	46.7084	20.2284	19.6006	6.07	11.6009	0	18272	152.63	0.2579
myciel5dy	39144	40.5647	79.1557	42.4718	28.86	16.2980	0	40383	197.31	3.1652
myciel6dy	108726	65.31727	59.49225	39.17426	94.96	14.58194	1	109942	25.18	1.1184
myciel7dy	411022	94.7880	106.6965	30.7758	377.08	8.7533	18	411022	193.94	0
queen5_5dy	21151	73.7655	29.5211	29.5211	4.76	29.5858	0	21151	70.41	0
queen6_6dy	39144	98.3058	50.1604	26.9977	284	20.5140	0	39676	207.02	1.3591
queen7_7dy	73264	154.6836	80.1393	45.1697	42.82	10.6983	0	73264	429.52	0
queen8_8dy	133130	190.5342	74.3438	33.4009	82.56	16.0512	2	133130	486.6	0
r125.1cdy	770623	203.5352	343.4828	58.9772	700	58.9772	120	770623	1788.57	0
r125.1dy	60931	85.06245	19.0019	16.75982	33.38	13.3765	0	61570	1315.53	1.0487
r125.5dy	501175	203.6558	214.9133	172.4574	1028.86	47.9300	26	501175	1786.26	0
spectraldy	15472	6.72	25.44	6.31	0.01	0	0	15472	0	0
treillisdy	1805.5625	0.02	129.23	113.11	0.03	33.1	0	1807.56	61	0.1106
turbocodedy	3195	33.49	84.32	20.09	0.13	20.09	0	3195	0.05	0
volterrady	178	7.87	7.87	7.87	0.01	7.87	0	178	0	0
zeroin_i1dy	486921	76.6642	277.5686	65.5265	1091.16	18.3601	79	490350	1792.83	0.7042
zeroin_i2dy	501989	86.4093	207.0074	39.6488	1086.34	11.0174	103	501989	1194.54	0
zeroin_i3dy	551001	81.0598	222.9487	47.8225	1063.72	12.5924	58	551001	1798.99	0

Table 7 is organized similarly. The first two columns are copied from Table 6. In the next six columns of Table 7, results of testing instances by using BVNS and RSVNS are given. These results appear in groups of three (value, running time (in seconds) and deviation value from the best known value (in percentage)). The individual results (solution value and running time) for each instance, for both algorithms proposed in Sánchez-Oro et. al [16], are provided by Jesús Sánchez-Oro, to whom we are very grateful. Deviation value is calculated as deviation from the Best known solution from the second column, in percentage. The last three columns are copied from Table 6, where the results of the presented VNS are given in the same way as the results for BVNS and RSVNS.

First, we want to point to the fact that with the introduced improved ILP formulation of Soto et. al from [11] and our ILP formulation for DMAP problem in embedded systems, CPLEX obtained better results than all considered methods for 10 instances. Therefore, solution presented in column "Best known" of Tables 6 and 7, is the best solution obtained in comparison with the results presented in Table 5 and solutions obtained by all considered methods (IM, CPA, GRASP, GRASP+EC, BVNS, RSVNS, and VNS).

Table 7: Comparison between new and the most recent methods from the literature for solving DMAP

		BVNS			RSVNS			VNS		
Instance	Best known	value	time	%dev	value	time	%dev	Obj.val.	time	%dev
adpcmdy	44192	44192	0.088	0	44192	0.146	0	44192	0	0
alidydy	107398	107766	100.004	0.3427	107846	100.022	0.4171	107398	180.05	0
cjpegdy	4466792	4466792	0.031	0	4466799	0.057	0.0002	4466800	0.01	0.0002
compressdy	342592	342592	0.004	0	342592	0.016	0	342592	0	0
fpsol2i2dy	2587875	2610611	100.225	0.8786	2587875	100.239	0	2605980	835.11	0.6996
fpsol2i3dy	2582988	2601944	100.205	0.7339	2582988	100.107	0	2616849	1000	1.3109
gsm_newdy	7808	7808	0.003	0	7808	0.019	0	7808	0	0
gsmdy	1355389.875	1355389.88	0.074	0	1355389.88	0.188	0	1355390	0.22	<0.0001
gsmdycorrdy	494118	494120	0.071	0.0004	494118	0.192	0	494118	1.85	0
inithx_i1dy	5652213	5652213	102.062	0	5654904	101.053	0.0476	5716414	1000	1.1359
lmsbdy	7409660	7409660	0.005	0	7409660	0.032	0	7409669	0	0.0001
lmsbv01dy	4350588	4350628	0.009	0.0009	4350588	0.046	0	4350640	0	0.0012
lmsbvdy	4323116	4327364	0.013	0.0983	4323116	0.045	0	4323294	0.03	0.0041
lmsbvdyexpdy	4366972	4367004	0.01	0.0007	4366972	0.052	0	4367024	0.01	0.0012
lpcdy	26888	26888	0.033	0	26888	0.16	0	26888	0.04	0
mpeg2enc2dy	9812	12463.2793	22.921	27.0208	12451.2793	39.915	26.8985	10915.28	0.23	11.2442
mpegdy	10613.625	10764.6875	2.573	1.4233	10694.375	4.167	0.7608	10648.88	480.44	0.3322
mug100_1dy	28890	30781	6.252	6.5455	30515	10.719	5.6248	30638	395	6.0505
mug100_25dy	28429	29409	6.058	3.4472	28997	10.629	1.9980	28876	323.07	1.5723
mug88_1dy	25305	26048	3.821	2.9362	25823	6.596	2.0470	25570	471.81	1.0472
mug88_25dy	24181	24886	3.75	2.9155	24569	6.487	1.6046	24365	392.18	0.7609
mulsol_i1dy	459598	465557	100.033	1.2966	459598	100.064	0	478739	1794.61	4.1647
mulsol_i2dy	552215	561626	100.001	1.7042	552215	100.006	0	557552	1792.89	0.9665
mulsol_i4dy	505927	511055	100.056	1.0136	506162	100.139	0.0464	505927	1785.29	0
mulsol_i5dy	518854	527097	100.056	1.5887	518854	100.046	0	520929	1797.09	0.3999
myciel3dy	6379	6382	0.015	0.0470	6379	0.048	0	6379	0.2	0
myciel4dy	18225	19041	0.172	4.4774	18678	0.405	2.4856	18272	152.63	0.2579
myciel5dy	39144	42324	0.992	8.1239	42105	1.763	7.5644	40383	197.31	3.1652
myciel6dy	108726	114466	14.774	5.2793	112502	25.107	3.4730	109942	25.18	1.1184
myciel7dy	411022	417987	100.002	1.6946	412993	100.058	0.4795	411022	193.94	0
queen5_5dy	21151	21903	0.182	3.5554	21763	0.404	2.8935	21151	70.41	0
queen6_6dy	39144	41018	1.223	4.7875	40722	2.062	4.0313	39676	207.02	1.3591
queen7_7dy	73264	75493	5.567	3.0424	75190	9.054	2.6288	73264	429.52	0
queen8_8dy	133130	140476	19.34	5.5179	137027	32.223	2.9272	133130	486.6	0
r125.1cdy	770623	854826	100.035	10.9266	836061	100.202	8.4916	770623	1788.57	0
r125.1dy	60931	62798	8.181	3.0641	62033	13.396	1.8086	61570	1315.53	1.0487
r125.5dy	501175	534619	100.012	6.6731	528754	100.101	5.5029	501175	1786.26	0
spectraldy	15472	15472	0.009	0	15472	0.032	0	15472	0	0
treillisdy	1805.5625	1869.375	0.411	3.5342	1829.8125	0.813	1.3431	1807.56	61	0.1106
turbocodedy	3195	3233	0.032	1.1894	3195	0.078	0	3195	0.05	0
volterrady	178	178	0.003	0	178	0.018	0	178	0	0
zeroin_i1dy	486921	494032	100.006	1.4604	486921	100.023	0	490350	1792.83	0.7042
zeroin_i2dy	501989	515845	100.004	2.7602	507314	100.098	1.0608	501989	1194.54	0
zeroin_i3dy	551001	572150	100.039	3.8383	564326	100.084	2.4183	551001	1798.99	0

Now, considering the results from Tables 6 and 7, none of the presented methods foundnd all best known solutions. For instance, CPLEX reached 18 of the best known solutions, of 31 solved. Further, IM, CPA, GRASP, and GRASP+EC together were successful in finding 8 of the best known, BVNS 10, RSVNS 20, and the presented VNS 19. Further more, the best known solution obtained only by IM was in two cases (the same number as for the BVNS), the best known solution obtained only by RSVNS was in 9 cases, while the best known solution obtained only by VNS was in 10 cases (same as with CPLEX). CPA, GRASP, and GRASP+EC together have no solution better than the solutions obtained by other considered methods for any of the tested instance. More details about this comparison are given in Table 8.

If we compare presented VNS with IM, CPA, GRASP, and GRASP+EC only (Table 6), we can see that: for 26 instances VNS finds better solutions than the solutions obtained with other four methods; for 12 instances VNS finds solutions equal to the best solutions obtained with other four methods; for 6 instances VNS finds worse solutions than the best solutions obtained with other four methods. Similarly, if we compare solutions of presented VNS with solutions obtained only with BVNS and RSVNS methods (Table 7), we can see that for 21 instances VNS finds better solutions than the solutions obtained with these two methods; for 9 instances solutions equal to the best solutions obtained with these two methods; for 14 instances VNS finds worse solutions than the best solutions obtained with these two methods.

Considering only results of the presented VNS, we can notice that running values are less than $1,800$s, which can be considered as very fast execution time. Comparing results of the presented VNS with results from the second column of Tables 6 and 7, we can notice that deviation values varies from $< 0.0001\%$ up to 11.2442% for instances where best known solutions are not reached. More precisely, with average deviation from the best solution value of 0.8513%, solution obtained by the proposed VNS differ from the best known solution in less than 1% for 14 instances, between 1% and 2% for 7 instances, and differ more than 2% for only 4 instances. Also, we can conclude that proposed VNS obtains the best improvements especially on very large instances ($T \geq 15, n \geq 20$ and $m \geq 5$). Notably, solutions for instances "r125.1cdy.col" and "r125.5dy.col" with respective parameters T, n, and m equal to $(75, 125, 23)$ and $(38, 125, 18)$, improved for 37.1% and 32.4%. Similarly, two instances of "zeroin" type with values parameters T, n, and m up to $(35, 211, 15)$ have the improvement of 10% and higher.

In Table 8 average objective function values (Avg.), average running time in seconds (Time(s)), average percentage deviation with respect to the best solution found (Dev(%)), and the number of times only that method matches the best result (#Best), are presented. Average value and average time for IM, CPA, GRASP, GRASP+EC are taken from [11]. Average value and average time for BVNS, RSVNS, and proposed VNS, together with the average deviation from the best known result, shown in the second column of Tables 6 and 7, are calculated in accordance with the data presented in Tables 6 and 7. The number of times a method matches the best result is calculated as the number of instances for which the best result is matched only by using that method. For instance, the best result for "mpeg2end2dy" and "mug100_1dy" was reached only by IM. In analogy to that, for 2 instances the best results were obtained only by the BVNS, for 9 instances only by the RSVNS, and for 10 instances the best results were obtained only by the proposed VNS. Given the fact that testings of considered methods were preformed on computers of different specifications, in order to compare running times of the proposed VNS algorithm with running times of the other considered algorithms, specifically, in order to compare the running times of the proposed VNS algorithm with the running times of the BVNS and RSVNS methods, a certain scaling factor should be applied. This scaling factor can be calculated as ratio between CPU mark (from https://www.cpubenchmark.net/compare.php) of computers performances. Since CPU of the computer used for testings of BVNS and RSVNS was marked as 4944 and CPU of ours as 1719, we can conclude that their computer is almost 3 times faster than ours. Regardless of the fact that average execution time of the proposed VNS algorithm is longer than the running time of other considered methods, longer execution time can be attributed to instances for which better objective function value is obtained. Now, as it can be seen from Table 8, although average value of the tested instances got using the proposed VNS method is the second best, the average deviation from the best known solution value obtained by the proposed VNS algorithm is the best. Indeed, solution value of the proposed VNS algorithm differ from the average value of the best known solutions in less than 1%. Further more, by using the proposed VNS algorithm for the biggest number of instances, the best known solution value is introduced.

Table 8: Comparisons between methods considered in this paper

	Avg.	Time (s)	Dev (%)	#Best
IM	1,378,609.24	300.72	76.81	2
CPA	1,375,151.61	300.48	477.73	0
GRASP	1,110,087.76	300.45	29.65	0
GRASP+EC	1,060,597.74	130.55	13.94	0
BVNS	1,006,790.25	34.08	2.77	2
RSVNS	1,003,751.30	35.62	1.97	9
VNS	1,004,086.27	539.78	0.85	10

In order to confirm the differences among the presented algorithm and algorithms known from the literature, we performed the Friedman non-parametric statistical test with all the individual values. The Friedman test ranks each algorithm in all instances according to the quality of the solution obtained, giving rank 1 to the best algorithm, 2 to the second one, and so on. If the averages differ greatly, the associated p-value or significance will be small. Similarly to the test presented in [16], the Friedman non-parametric statistical test is preformed for IM, GRASP+EC, BVNS, RSVNS, and VNS. The resulting p-value (lower than 0.001) obtained in this experiment clearly indicates that there are statistically significant differences among the tested methods. More precisely, the average rank values produced by this test are 3.9886 (IM), 4.0682 (GRASP+EC), 2.9091 (BVNS), 2.0227 (RSVNS), and 2.0114 (VNS), but additional p information shows that VNS and RSVNS statistically significantly differ from IM, GRASP, GRASP+EC, and BVNS but not from each other.

We conducted an additional statistical test (Wilcoxon signed rank test) to perform pair-wise comparisons between our method (VNS) and the previous algorithms (IM, GRASP + EC, BVNS and RSVNS). Table 9 presents the results of Wilcoxon signed rank test. Since *p*-value obtained in the first three tests, presented in Table 9, is lower than 0.001, it means that there are significant statistical differences between the compared algorithms. Therefore, VNS clearly outperforms the results obtained by IM, GRASP+EC, and BVNS. Further, since *p*-value obtained in the fourth test is much higher than the significant, we can consider these methods as the methods of the similar quality. Still, given that with the proposed VNS method better solution was obtained for 21 instance, while with RSVNS better solution was obtained for 14 instances, we give VNS a little advantage.

Table 9: Results of the Wilcoxon signed rank test run over each pair of algorithms of the final experiment

$A1$	$A2$	$A1 < A2$	$A1 > A2$	$A1 = A2$	p-value
VNS	IM	28	5	11	≤ 0.001
VNS	GRASP+EC	37	2	5	≤ 0.001
VNS	BVNS	30	8	6	≤ 0.001
VNS	RSVNS	21	14	9	0.2318

6. CONCLUSIONS

This paper presents our improvement of the existing ILP formulation for DMAP. The proposed cost function covers all oversights we referred to and therefore, with the proposed improvements, our new ILP formulation corresponds to the DMAP completely. In addition, we presented a new ILP formulation and a new metaheuristic approach based on the Variable Neighborhood Search. In order to compare the results of the proposed VNS heuristic and the methods previously established, we tested all of them on the same set of instances. From the computational results we can conclude that the proposed VNS heuristic effectively solves DMAP, providing (including CPLEX solutions of the new ILP formulation) 20 new best known solution values. Results obtained by the VNS are better than the results obtained by each method considered individually and better or equal to the results obtained by all methods previously established. Indeed, results obtained by the VNS are better or equal to the best result obtained by IM, CPA, GRASP, GRASP+EC, BVNS and RSVNS, combined for 25 instances. Execution time could not be easily compared given that almost all of the proposed methods for solving DMAP were tested on different types of computers, but all solutions were reached in a reasonable time limit or even less than $1,800$ seconds.

Building similar dynamic memory allocation problems in embedded systems such as problems which involve calculation memory, possible movings during time blocks, conflicts which involve three or more data structures at the same time could be considered as the future work. Also, implementation of parallelization into VNS heuristic can be considered as well.

Acknowledgments: The authors are grateful to anonymous referees for their remarks and suggestions that helped to improve the manuscript. This research was supported by the Research Grants 174010 and TR36006 of the Serbia Ministry of Education, Science and Technological Developments.

REFERENCES

[1] Atienza, D., Mamagkakis, S., Poletti, F, Mendias, J. M., Catthoor, F., Benini, L., Soudris, D., "Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems", *INTEGRATION, the VLSI journal*, 39 (2) (2006) 113–130.

[2] Hanafi, S., Lazić, J., Mladenović, N., Wilbaut, C., Crévits, I., "New variable neighbourhood search based 0-1 MIP heuristics", *Yugoslav Journal of Operations Research*, 25 (3) (2015) 343-360.

[3] Hansen, P., Mladenović, N., "An introduction to variable neighborhood search", *Meta-heuristics*, Springer, Boston, MA, 1999, 433-458.

[4] Hansen, P., Mladenović, N., Pérez, J.A.M,. "Variable neighbourhood search: methods and applications", *Annals of Operations Research*, 175 (1) (2010) 367-407.

[5] Hansen, P., Mladenović, N., "Variable neighborhood search: Principles and applications", *European journal of operational research*, 130 (3) (2001) 449–467.

[6] Hansen, P., Mladenović, N., *Developments of variable neighborhood search*, Essays and surveys in metaheuristics, Springer, 2002, 415–439.

[7] Hansen, P., Mladenović, N., *Variable neighbourhood search*, Handbook of Metaheuristics, Springer, Boston, MA, 2003, 145-184.

[8] Ivanović, M., Dugošija, D., Savić, A., Urošević, D., "A new integer linear formulation for a memory allocation problem", in proc. *Balcor 2013, XI Balcan Conference on Operational Research*, 2013, 284 288.

[9] Mladenović, N., Hansen, P., "Variable neighborhood search", *Computers & Operations Research*, 24 (11) (1997) 1097–1100.

[10] Mladenović, N., "A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization", *Optimization Weeks*, 112 (1995) 112-112.

[11] Sevaux, M., Rossi, A., Soto, M., Duarte, A., Martí, R., "Grasp with ejection chains for the dynamic memory allocation in embedded systems", *Soft Computing*, 18 (8) (2014) 1515–1527.

[12] Soto, M., Rossi, A., Sevaux, M., "A mathematical model and a metaheuristic approach for a memory allocation problem", *Journal of Heuristics*, 18 (1) (2012) 149–167.

[13] Soto, M., Rossi, A., Sevaux, M., "Two iterative metaheuristic approaches to dynamic memory allocation for embedded systems", *European Conference on Evolutionary Computation in Combinatorial Optimization*, Springer, Berlin, Heidelberg, 2011, 250–261.

[14] Soto, M., Sevaux, M., Rossi, A., Laurent, J., *Memory Allocation Problems in Embedded Systems: Optimization Methods*, John Wiley & Sons, Inc, Hoboken, NJ, 2013.

[15] Sánchez-Oro, J., Sevaux, M., Rosi, A., Martí, R., Duarte, A., "Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies", *Electronic Notes in Discrete Mathematics*, 47 (2015) 85–92.

[16] Sánchez-Oro, J., Sevaux, M., Rosi, A., Martí, R., Duarte, A., "Improving the performance of embedded systems with variable neighborhood search", *Applied Soft Computing*, Elsevier, 53 (2017) 217-226.

[17] Wuytack, S., Catthoor, F., Nachtergaele, L., De Man, H., "Power exploration for data dominated video applications", in: *Proceedings of the 1996 international symposium on Low power electronics and design*, IEEE Press, 1996, 359–364.

Engineering and Applied Science Letters

PSR Press

Article

Dominator and total dominator colorings in vague graphs

Lian Chen[1], Huiqin Jiang[2], Zehui Shao[1,*] and Marija Ivanović[3]

[1] Institute of Computing Science and Technology, Guangzhou University, Guangzhou 510006, China.; chenlian@gzhu.edu.cn(L.C); zshao@gzhu.edu.cn(Z.S)

[2] School of Information Science and Engineering, Chengdu University, Chengdu 610106, China.; hq.jiang@hotmail.com(H.J)

[3] Faculty of Mathematics, University of Belgrade, Studentski trg 16/IV, 11 000 Belgrade, Serbia.; maria.ivanovic@gmail.com(M.I)

* Correspondence: zshao@gzhu.edu.cn

Received: 1 March 2019; Accepted: 15 May 2019; Published: 28 May 2019.

Abstract: The concept of vague graph was introduced early by Ramakrishna and substantial graph parameters on vague graphs were proposed such graph coloring, connectivity, dominating set, independent set, total dominating number and independent dominating number. In this paper, we introduce the concept of the dominator coloring and total dominator coloring of a vague graph and establish mathematical modelling for these problems.

Keywords: Dominator coloring, fuzzy graph theory, total dominator coloring, vague graph.

1. Introduction

Fuzzy set generalize classical sets by use of a membership function such that each element is assigned a number in the real unit interval [0,1], which measures its grade of membership in the set. The theory of fuzzy sets was proposed by Zadeh in 1965 [1]. Since then, the theory was used in a wide range of domains in which information is incomplete or imprecise, such as such as management science, medical science, social science, financial science, environment science and bioinformatics [2]. In 1993, Gau *et al.* [3] presented the concept of vague set theory as a generalization of fuzzy set theory, which allow a separation of evidence for membership (grade of membership) and evidence against membership (negation of membership). They used a subinterval of [0,1] to replace the value of an element in a set. That is, a vague set is characterized by two functions. Namely, a truth-membership function $t_v(x)$ and false-membership function $f_v(x)$ are used to describe the boundaries of the membership degree.

Graph theory is a very useful and well developed branch of discrete mathematics, and it also is an important tool for modeling many types of relations and processes in biological, physical, social and information systems. Realizing the importance of graph theory and inspiring of Zadeh's fuzzy relations [4], Kauffman [5] proposed the definition of fuzzy graph in 1973. Then Rosenfeld [6] proposed another elaborated definition of fuzzy graph in 1975. Since then, there was a vast research on fuzzy graph [7–19]. Inspired by fuzzy graph, in 2009, Ramakrishna [20] introduced the concept of vague graphs and studied some important properties. After that, Samata *et al.* [21] analysed the concepts of vague graphs and its strength. Rashmanlou *et al.* [22] introduced the notion of vague h-morphism on vague graphs and regular vague graphs, and they investigated some properties of an edge regular vague graph [23]. At the same time, they introduced some connectivity concepts in the vague graphs [24].

The Dominator coloring of a graph was proposed by Gera et al [25] in 2006. In the same paper, they showed that dominator chromatic number is NP-complete. After that, they studied the bounds and realization of the dominator chromatic number in terms of chromatic number and domination number [26] and the dominator colorings in bipartite graphs [27]. Recently, several researchers have theoretically investigated the dominator coloring number of Claw-free graph [28], Certain Cartesian Products [29], trees [30] and more [31,32]. Motivated by dominator chromatic number, Kazemi [33] studied the new concept of a total dominator chromatic number of a graph. And they showed that total dominator chromatic number is NP-complete. A survey of total dominator chromatic number in graphs can also be found in [34,35].

Borzooei *et al.* [36] in their work introduced the concepts of special kinds of dominating sets in vague graph. Kumar *et al.* [37] discuss the new concepts of coloring in vague graphs with application. In this paper, we introduce the concept of the dominator coloring and total dominator coloring of a vague graph and establish mathematical modelling for these problems.

2. Preliminaries

A vague set A in an ordinary finite non-empty set X is a pair (t_A, f_A), where $t_A : X \rightarrowtail [0,1]$, $f_A : X \rightarrowtail [0,1]$, and $0 \leq t_A(x) + f_A(x) \leq 1$ for each element $x \in X$. Note that the truth-membership $t_A(x)$ is considered as the lower bound on grade of membership of x derived from the evidence for $x \in X$ and the false-membership $f_A(x)$ is the lower bound on negation of membership of x derived from the evidence against $x \in X$. The grade of membership for x is characterized by the interval $[t_A(x), 1 - f_A(x)]$ not a crisp value. And if $t_A(x) = 1 - f_A(x)$ for all $x \in X$, the vague set degrades to a fuzzy set.

In this paper, we denote by P_n, C_n, K_n the path, cycle and complete graph on n vertices, respectively. The complete bipartite graph with part size m, n is denoted by $K_{m,n}$ and the ladder graph is the Cartesion product of P_2 and C_n, denoted by $P_2 \square C_n$.

Definition 1. Let $G = (V, E)$ be a graph. A pair $G' = (A, B)$ is called a vague graph on G where $A = (t_A, f_A)$ is a vague set on V and $B = (t_B, f_B)$ is a vague set on E such that $t_B(uv) \leq min\{t_A(u), t_A(v)\}$, $f_B(uv) \geq max\{f_A(u), f_A(v)\}$ for each $uv \in E$.

Definition 2. For a vague graph $G = (A, B)$, an edge uv is called a strong edge if $t_B(uv) = min\{t_A(u), t_A(v)\}$, $f_B(uv) = max\{f_A(u), f_A(v)\}$. Let $N(u) = \{v | uv \text{ is a strong edge in } G\}$ and $N[u] = N(u) \cup \{u\}$.

We say u dominates all vertices in $N(u)$ and totally dominates all vertices in $N[u]$.

Definition 3. Dominator coloring of a vague graph G is a coloring of the vertices of G such that every vertex dominates all vertices of at least one other class. The dominator chromatic number $\chi^d(G)$ of G is the minimum number of colors among all dominator colorings of G.

Definition 4. Total dominator coloring of a vague graph G is a coloring of the vertices of G such that every vertex totally dominates all vertices of at least one other class. The total dominator chromatic number $\chi_t^d(G)$ of G is the minimum number of colors among all total dominator colorings of G.

3. Dominator coloring problems

Let $[k] = \{1, 2, \ldots, k\}$. Let $V_c \subseteq V$ denotes set of vertices with assigned color c. Further, let decision variables $x_{i,c}$ be defined as

$$x_{i,c} = \begin{cases} 1, & i \in V_c \\ 0, & i \in V_c \end{cases}$$

For a vague graph G and an integer k, let E^s be the set of all strong edges of G. We propose integer linear programming (ILP) formulations (called Dominator Coloring ILP and Total Dominator Coloring ILP, respectively), for the dominator coloring problem and total dominator coloring problem as follows:

Dominator coloring ILP

$$\sum_{c=1}^{k} x_{i,c} = 1, \quad i \in V \tag{1}$$

$$\sum_{i \in V}^{k} x_{i,c} \geq 1, \quad c \in [k] \tag{2}$$

$$x_{i,c_1} + x_{j,c_2} + M_{i,c_1,c_2} \leq 2, \quad c_1, c_2 \in [k], i \in V(G), j \in V \backslash N[i] \tag{3}$$

$$\sum_{c_2=1}^{k} M_{i,c_1,c_2} \geq 1, \quad c_1 \in [k], i \in V \tag{4}$$

$$x_{i,c} + x_{j,c} \leq 1, \quad c \in [k], (i,j) \in E^s \tag{5}$$

$$x_{i,c} \in \{0,1\}, \quad c \in [k], i \in V \tag{6}$$

$$M_{i,c_1,c_2} \in \{0,1\}, \quad \{c_1, c_2\} \subseteq [k], i \in V \tag{7}$$

Theorem 5. *Conditions* $(1)-(7)$ *defined for the graph G are satisfied if and only if G admits a dominator coloring with k colors.*

Proof. Condition (1) ensures that each vertex is assigned with exactly one color. Condition (2) ensures that each color should be used. Conditions (3) and (4) ensure that every vertex dominates all vertices of at least one other class. Condition (5) ensures that the assignment is a proper coloring. Conditions (6) and (7) ensure that each variable is boolean. Therefore, if each condition is satisfied, then G admits a dominator coloring with k colors.

$\Leftarrow$ By the definition of the dominator coloring, it is clear that Conditions $(1)-(7)$ defined for the graph G are satisfied. □

Total dominator coloring ILP

$$\sum_{c=1}^{k} x_{i,c} = 1, \quad i \in V \tag{8}$$

$$\sum_{i \in V}^{k} x_{i,c} \geq 1, \quad c \in [k] \tag{9}$$

$$x_{i,c_1} + x_{j,c_2} + M_{i,c_1,c_2} \leq 2, \quad \{c_1, c_2\} \in [k], i \in V(G), j \in V \backslash N[i] \tag{10}$$

$$\sum_{c_2 \neq c_1, c_2=1}^{k} M_{i,c_1,c_2} \geq 1, \quad c_1 \in [k], i \in V \tag{11}$$

$$x_{i,c} + x_{j,c} \leq 1, \quad c \in [k], (i,j) \in E^s \tag{12}$$

$$x_{i,c} \in \{0,1\}, \quad c \in [k], i \in V \tag{13}$$

$$M_{i,c_1,c_2} \in \{0,1\}, \quad \{c_1, c_2\} \subseteq [k], i \in V \tag{14}$$

Theorem 6. *Conditions* $(8)-(14)$ *defined for the graph G are satisfied if and only if G admits a total dominator coloring with k colors.*

Proof. Condition (8) ensures that each vertex is assigned with exactly one color. Condition (9) ensures that each color should be used. Conditions (10) and (11) ensure that every vertex totally dominates all vertices of at least one other class. Condition (12) ensures that the assignment is a proper coloring. Conditions (13) and (14) ensure that each variable is boolean. Therefore, if each condition is satisfied, then G admits a total dominator coloring with k colors.

$\Leftarrow$ By the definition of the total dominator coloring, it is clear that Conditions $(1)-(7)$ defined for the graph G are satisfied. □

Example 1. Let G be a vague graph depicted in Figure 1. Then the set of strong edges is $\{(u_1u_2), (u_2v_2), (u_2u_4), (u_2u_3), (u_3u_4), (u_4v_3), (u_4u_5), (u_5v_4), (u_5u_6), (u_6u_7), (v_1v_2), (v_2v_3), (v_3v_4), (v_4v_5)\}$.

Example 2. Let G be a vague graph depicted in Figure 2. Then by solving the instance from Dominator Coloring ILP, we obtain $\gamma^d(G) = 6$. A dominator coloring f with 6 colors is $f(u_1) = 1$, $f(u_2) = 2$, $f(u_3) = 4$, $f(u_4) = 1, f(u_5) = 4, f(u_6) = 6$, $f(u_7) = 1, f(v_1) = 3$, $f(v_2) = 1$, $f(v_3) = 4$, $f(v_4) = 5$, $f(v_5) = 1$ which is presented in Figure 2.

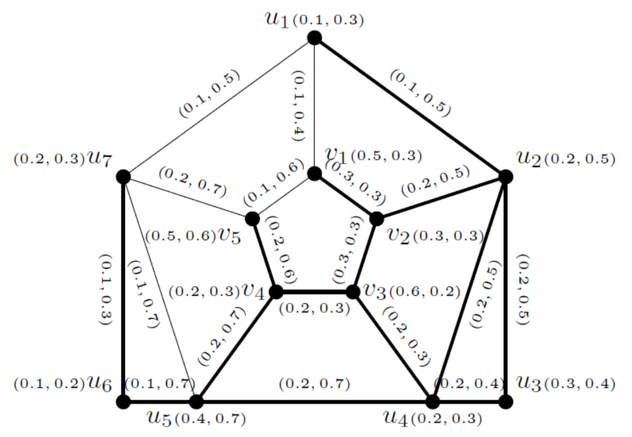


Figure 1. An example of a vague graph G

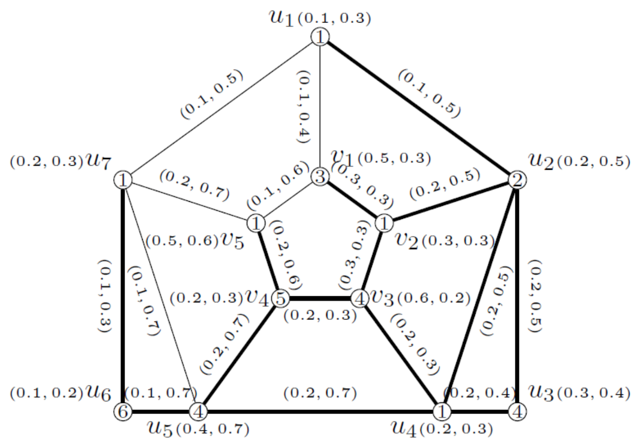


Figure 2. A dominator coloring of G with 6 colors

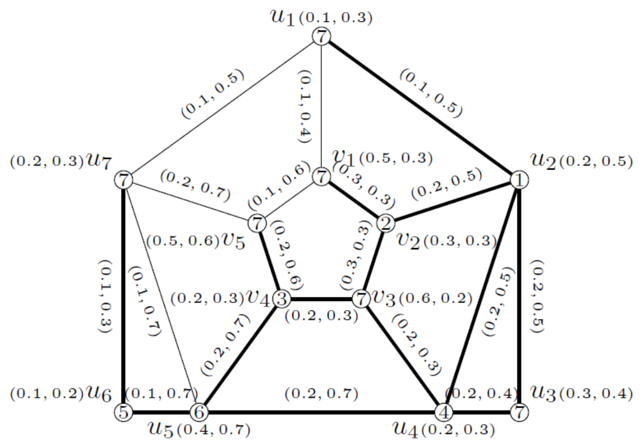


Figure 3. A total dominator coloring of G with 7 colors

Example 3. Let G be a vague graph depicted in Figure 3. Then by solving the instance from Total Dominator Coloring ILP, we obtain $\gamma_t^d(G) = 7$. A dominator coloring f with 7 colors is $f(u_1) = 7,\ f(u_2) = 1, f(u_3) = 7, f(u_4) = 4,\ f(u_5) = 6, f(u_6) = 5, f(u_7) = 7,\ f(v_1) = 7,\ f(v_2) = 2, f(v_3) = 7,\ f(v_4) = 3, f(v_5) = 7$ which is presented in Figure 3.

4. Dominator coloring number of some classes of vague graphs

Definition 7. For a vague graph G, we define an underlying graph of G, denoted by $\widetilde{G}$, with $V(\widetilde{G}) = V(G)$, and $xy \in E(\widetilde{G})$ if and only if $x \in N(y)$ in G.

By the definition of Dominator coloring, we have

Proposition 8. *For any vague graph G, $\chi^d(G) = \chi^d(\widetilde{G})$ and $\chi_d^t(G) = \chi_d^t(\widetilde{G})$.*

Proposition 9. *(see [28]) For any vague graph G with $\widetilde{G} \cong K_n$, we have $\chi_d(G) = \chi_d^t(G) = n$.*

Proof. By the definition, we have $\chi_d^t(G) \geq \chi_d(G) \geq \chi(G) = n$. Let $V(G) = \{v_1, v_2, \ldots, v_n\}$. We consider a function $f : V(G) \rightarrowtail \{1, 2, \ldots, n\}$ with $f(v_i) = i$ for any i, then we have f is a total dominator coloring of G with n colors. Therefore, we have $\chi_d^t(G) \leq n$ and so the desired result holds. □

The following results are straightforward:

Proposition 10. *(see [28]) For any vague graph G with $\widetilde{G} \cong K_{m,n}$, we have $\chi^d(G) = \chi_d^t(G) = 2$.*

Proposition 11. *(see [28]) For any vague graph G with $\widetilde{G} \cong C_n (n \geq 3)$, we have $\chi^d(G) = 3$ for $n \equiv 3$ (mod 6) and $\chi_d(G) = 2$ otherwise.*

Proposition 12. *(see [35]) For any vague graph G with $\widetilde{G} \cong P_n$ or $C_n (n \geq 3)$, we have $\chi_d^t(G) = \left[\frac{n}{2}\right] + \left[\frac{n}{4}\right] - \left[\frac{n}{4}\right]$.*

The Cartesian product $G \square H$ of two graphs G and H is a graph with $V(G) \times V(H)$ and two vertices (g_1, h_1) and (g_2, h_2) are adjacent if and only if either $g_1 = g_2$ and $(h_1, h_2) \in E(H)$, or $h_1 = h_2$ and $(g_1, g_2) \in E(G)$. Let $V(C_n) = \{1, 2, 3, \ldots n\}$, $E(C_n) = \{i(i+1)\}$, $1n|i = 1, 2, \ldots n-1\}$ and $V(P_2) = \{1, 2\}$, $E(P_2) = \{(12)\}$. Let $u_{i,j}$ be a vertex of $P_2 \square C_n$ where $i = 1, 2$, $j = 1, 2, \ldots n$. We have the following result:

Proposition 13. *For any vague graph G with $\widetilde{G} \cong P_2 \square C_n$ with $n \geq 6$,*

$$\gamma_t^d(G) \leq \begin{cases} \frac{2n}{3} + 2, & n \equiv 0 \quad (mod\ 6) \\ \frac{2n}{3} + 4, & n \equiv 1, 2\ (mod\ 6) \\ \frac{2n}{3} + 3, & n \equiv 3 \quad (mod\ 6) \\ \frac{2n}{3} + 2, & n \equiv 4, 5\ (mod\ 6) \end{cases}$$

Proof. We use two lines of numbers to denote a total dominator coloring of $P_2 \square C_n$. The total dominator coloring can be represented as a $2 \times n$ array as follows:

$$f(P_2 \square C_n) = \begin{cases} f(u_{1,1})\ f(u_{1,2}) \ldots f(u_{1,n-1})\ f(u_{1,n}) \\ f(u_{2,1})\ f(u_{2,2}) \ldots f(u_{2,n-1})\ f(u_{2,n}) \end{cases}$$

If $i = 1$ and $j \equiv 1, 3$ (mod 6), let $f(u_{i,j}) = 1$.
If $i = 1$ and $j \equiv 2, 4$ (mod 6), let $f(u_{i,j}) = 2$.
If $i = 1$ and $j \equiv 5$ (mod 6), let $f(u_{i,j}) = 4\frac{j}{6} + 5$.
If $i = 1$ and $j \equiv 0$ (mod 6), let $f(u_{i,j}) = 4\frac{j}{6} + 6$.
If $i = 2$ and $j \equiv 4, 5$ (mod 6), let $f(u_{i,j}) = 1$.
If $i = 2$ and $j \equiv 0, 5$ (mod 6), let $f(u_{i,j}) = 2$.
If $i = 2$ and $j \equiv 2$ (mod 6), let $f(u_{i,j}) = 4\frac{j}{6} + 3$.
If $i = 2$ and $j \equiv 3$ (mod 6), let $f(u_{i,j}) = 4\frac{j}{6} + 4$.
We will consider the following cases:

Case 1. $n \equiv 0 \pmod 6$.
Obviously, f is total dominator coloring with desired number of colors. For example, let $n = 12$, we have

$$f(P_2 \square C_{12}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ 2\ 9\ 10 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1 \end{cases}$$

Case 2. $n \equiv 1, 2, 4, 5 \pmod 6$.
Let $h(x) = f(x)$ for any $x \in V(P_2 \square C_n) \backslash \{u_{1,n}, u_{2,n}\}$, $h(u_{1,n}) = 2 \times \frac{n}{3}$, $h(u_{2,n}) = 2 \times \frac{n}{3} + 4$. Obviously, h is total dominator coloring with desired number of colors. For example, let $n = 13$, we have

$$f(P_2 \square C_{13}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ 2\ 9\ 10\ 11 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1\ 12 \end{cases}$$

Let $n = 14$, we have

$$f(P_2 \square C_{14}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ 2\ 9\ 10\ 1\ 11 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1\ 2\ 12 \end{cases}$$

Let $n = 16$, we have

$$f(P_2 \square C_{16}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ 2\ 9\ 10\ 1\ 2\ 1\ 13 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1\ 2\ 11\ 12\ 14 \end{cases}$$

Let $n = 17$, we have

$$f(P_2 \square C_{17}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ \ 2\ 9\ 10\ 1\ 2\ 1\ 2\ 13 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1\ 2\ 11\ 12\ 1\ 14 \end{cases}$$

Case 3. $n \equiv 3 \pmod 6$.
Let $h(x) = f(x)$ for any $x \in V(P_2 \square C_n) \backslash \{u_{1,n}\}$, $h(u_{1,n}) = \frac{2n}{3} + 3$. Obviously, h is total dominator coloring with desired number of colors. As an example, let $n = 15$, we have

$$f(P_2 \square C_{15}) = \begin{cases} 1\ 2\ 1\ 2\ 5\ 6\ 1\ 2\ 1\ 2\ 9\ 10\ 1\ 2\ 13 \\ 2\ 3\ 4\ 1\ 2\ 1\ 2\ 7\ 8\ 1\ 2\ 1\ 2\ 11\ 12 \end{cases}$$

Now the proof is complete. $\square$

5. Conclusion

Fuzzy graph theory has substantial applications for real-world life in different domains, such as in the fields of biological science, neural networks, decision making, physics and chemistry. At present, the graph coloring problem can be applied in sequencing, timetabling, scheduling, electronic bandwidth allocation, computer register allocation and printed circuit board testing. Also the domination is also one of the fundamental concepts in graph theory and it has been wide used to distributed computing, biological networks, resource allocation and social networks. In this paper, motivated with the combination of fuzzy graph theory, graph coloring and graph domination, we introduce the concept of the dominator coloring and total dominator coloring of a vague graph and establish mathematical modelling for these problems.

Author Contributions: All authors contributed equally to the writing of this paper. All authors read and approved the final manuscript.

Conflicts of Interest: "The authors declare no conflict of interest."

References

[1] Zadeh, L. A. (1965). Fuzzy sets. *Information and control, 8*(3), 338-353.
[2] Pappis, C. P., Siettos, C. I., & Dasaklis, T. K. (2013). Fuzzy sets, systems, and applications. *Encyclopedia of Operations Research and Management Science*, 609-620.
[3] Gau, W. L., & Buehrer, D. J. (1993). Vague sets. *IEEE transactions on systems, man, and cybernetics, 23*(2), 610-614.
[4] Zadeh, L. A. (1971). Similarity relations and fuzzy orderings. *Information sciences, 3*(2), 177-200.

[5] Kaufmann, A. (1975). *Introduction to the theory of fuzzy subsets* (Vol. 2). Academic Pr.

[6] Zadeh, L. A. (1977). Fuzzy sets and their application to pattern classification and clustering analysis. In *Classification and clustering* (pp. 251-299). Academic press.

[7] Samanta, S., Pal, M., Rashmanlou, H., & Borzooei, R. A. (2016). Vague graphs and strengths. *Journal of Intelligent & Fuzzy Systems, 30*(6), 3675-3680.

[8] Borzooei, R. A., & Rashmanlou, H. (2015). Ring sum in product intuitionistic fuzzy graphs. *Journal of advanced research in pure mathematics, 7*(1), 16-31.

[9] Jun, Y. B. (2006). Intuitionistic fuzzy subsemigroups and subgroups associated by intuitionistic fuzzy graphs. *Communications of the Korean Mathematical Society, 21*(3), 587-593.

[10] Rashmanlou, H., Samanta, S., Pal, M., & Borzooei, R. A. (2015). A study on bipolar fuzzy graphs. *Journal of Intelligent & Fuzzy Systems, 28*(2), 571-580.

[11] Rashmanlou, H., Samanta, S., Pal, M., & Borzooei, R. A. (2015). Bipolar fuzzy graphs with categorical properties. *International Journal of Computational Intelligence Systems, 8*(5), 808-818.

[12] Rashmanlou, H., & Jun, Y. B. (2013). Complete interval-valued fuzzy graphs. *Annals of Fuzzy Mathematics and Informatics, 6*(3), 677-687.

[13] Samanta, S., & Pal, M. (2011). Fuzzy tolerance graphs. *International Journal of Latest Trends in Mathematics, 1*(2), 57-67.

[14] Samanta, S., & Pal, M. (2011). Fuzzy threshold graphs. *CIIT International Journal of Fuzzy Systems, 3*(12), 360-364.

[15] Sunitha, M. S., & Vijayakumar, A. (2002). *Complement of a fuzzy graph. Indian Journal of pure and applied Mathematics, 33*(9), 1451-1464.

[16] Samanta, S., & Pal, M. (2015). Fuzzy planar graphs. *IEEE Transactions on Fuzzy Systems, 23*(6), 1936-1942.

[17] Samanta, S., Akram, M., & Pal, M. (2015). M-Step fuzzy copetition graphs. *Journal of Applied Mathematics and Computing, 47*(1-2), 461-472.

[18] Samanta, S., & Pal, M. (2013). Fuzzy k-competition graphs and p-competition fuzzy graphs. *Fuzzy Information and Engineering, 5*(2), 191-204.

[19] Samanta, S., Pal, M., & Pal, A. (2014). New concepts of fuzzy planar graph. *International Journal of Advanced Research in Artificial Intelligence, 3*(1), 52-59.

[20] Ramakrishna, N. (2009). Vague graphs. *International Journal of Computational Cognition, 7*(51-58), 19.

[21] Samanta, S., Pal, M., Rashmanlou, H., & Borzooei, R. A. (2016). Vague graphs and strengths. *Journal of Intelligent & Fuzzy Systems, 30*(6), 3675-3680.

[22] Rashmanlou, H., Samanta, S., Pal, M., & Borzooei, R. A. (2016). A study on vague graphs. *SpringerPlus, 5*(1), 1234.

[23] Borzooei, R. A., Rashmanlou, H., Samanta, S., & Pal, M. (2016). Regularity of vague graphs. *Journal of Intelligent & Fuzzy Systems, 30*(6), 3681-3689.

[24] Rashmanlou, H., & Borzooei, R. A. (2016). Vague graphs with application. *Journal of Intelligent & Fuzzy Systems, 30*(6), 3291-3299.

[25] Gera, R., Rasmussen, C. W., & Horton, S. (2006). Dominator colorings and safe clique partitions. *Faculty Publications 181*(7) (2006), 19-32. **Need to cross check this reference**

[26] Gera, R. (2007). On dominator colorings in graphs. *Graph Theory Notes of New York, 52*, 25-30.

[27] Gera, R. (2007, April). On the dominator colorings in bipartite graphs. In *Fourth International Conference on Information Technology (ITNG'07)* (pp. 947-952). IEEE.

[28] Abdolghafurian, A., Akbari, S., Ghorban, S. H., & Qajar, S. (2014). Dominating Coloring Number of Claw-free Graphs. *Electronic Notes in Discrete Mathematics, 45*, 91-97.

[29] Chen, Q., Zhao, C., & Zhao, M. (2017). Dominator colorings of certain cartesian products of paths and cycles. *Graphs and Combinatorics, 33*(1), 73-83.

[30] Merouane, H., & Chellali, M. (2012). On the dominator colorings in trees. *Discussiones Mathematicae Graph Theory, 32*(4), 677-683.

[31] Chellali, M., & Maffray, F. (2012). Dominator colorings in some classes of graphs. *Graphs and Combinatorics, 28*(1), 97-107.

[32] Bagan, G., Boumediene-Merouane, H., Haddad, M., & Kheddouci, H. (2017). On some domination colorings of graphs. *Discrete Applied Mathematics, 230*, 34-50.

[33] Kazemi, A. P. (2015). Totat Dominator Chromatic number of a Graph. *Transactions on Combinatorics* 4(2), 57-68.

[34] Kazemi, A. P. (2014). Total dominator coloring in product graphs. *Util. Math, 94*, 329-345.

[35] Henning, M. A. (2015). Total dominator colorings and total domination in graphs. *Graphs and Combinatorics, 31*(4), 953-974.

[36] Borzooei, R. A., & Rashmanlou, H. (2015). Domination in vague graphs and its applications. *Journal of Intelligent & Fuzzy Systems 29*(5), 1933-1940.

[37] Kishore Kumar, P. K., Lavanya, S., Broumi, S., & Rashmanlou, H. (2017). New concepts of coloring in vague graphs with application. *Journal of Intelligent & Fuzzy Systems*, *33*(3), 1715-1721.

A NEW LINEAR–TIME ALGORITHM FOR COMPUTING THE WEAK ROMAN DOMINATION NUMBER OF A BLOCK GRAPH

Marija Ivanovic
Institute of Physics Belgrade
7 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Dragan Urosevic
Mathematical Institute of the Serbian Academy of Sciences and Arts
64 PUBLICATIONS **765** CITATIONS
SEE PROFILE

A NEW LINEAR-TIME ALGORITHM FOR COMPUTING THE WEAK ROMAN DOMINATION NUMBER OF A BLOCK GRAPH

MARIJA IVANOVIĆ[1], DRAGAN UROŠEVIĆ[2]

[1] University of Belgrade, Faculty of Mathemaics, marijai@math.rs

[2] Mathematical Institute, Belgrade, draganu@mi.sanu.ac.rs

Abstract: *In this paper we show that the known linear-time algorithm for solving the weak Roman domination problem on a block graph, from the literature, does not always find a weak Roman Domination function (WRDF) of minimal total weight. Furthermore, we present our newly developed linear-time algorithm that finds a WRDF of minimal total weight for the block graph.*

Keywords: *weak Roman domination number, block graph, linear time algorithm*

1. INTRODUCTION

The Roman domination problem starts with the assumption that the Roman Empire can be represented as a graph such that every vertex represents a province. Two vertices are adjacent if the corresponding provinces are neighbors, or there is direct connection between them, allowing fast traveling from one province to another. Assuming that a province is safe from the attack if at least one legion is stationed in it and that the unsafe province can be defended if it has a neighbor with two stationed legions, Stewart (1999) initiated a new variant of the domination problem named *Roman domination problem*. Since the Roman domination problem was defined by Stewart and ReVelle and Rosing (Stewart (1999); ReVelle and Rosing (2000)), many articles have been published (i.e. Dreyer (2000); Henning (2002); Cockayne et al. (2004); Chambers et al. (2009); Liu and Chang (2013); Ahangar et al. (2014); Beeler et al. (2016)). In this paper we are focused on one of its variant, named *weak Roman domination problem*. The weak Roman domination problem, introduced by Henning and Hedetniemi (2003), can be described as follows.

Let $G = (V,E)$ be a graph, $f : V \to \{0,1,2\}$ a function, and let $f(u)$ denote the weight of a vertex $u \in V$. A vertex u with $f(u) = 0$ is undefended if it is not adjacent to the neighbor with positive weight. The function f is called a *weak Roman dominating function* (WRDF) if every vertex u with $f(u) = 0$ is adjacent to a vertex v with $f(v) > 0$ such that the function $f' : V \to \{0,1,2\}$, defined as $f'(u) = 1$, $f'(v) = f(v) - 1$ and $f'(w) = f(w)$ when $w \in V \setminus \{u,v\}$, has no undefended vertices. Assuming that $f(u)$ is equal to a number of legions assigned to the province represented by the vertex u, the number of legions assigned to graph G, with respect to the function f, is called *total function weight* and is calculated as $w(f) = \sum_{u\in V} f(u)$. The *weak Roman domination number*, denoted by $\gamma_r(G)$, is equal to the minimal total weight of all possible WRDF that can be defined for that graph, G ($\gamma_r(G) = \min\{w(f)|f \text{ is a WRDF for } G\}$). A WRDF for graph G of weight γ_r is called γ_r-function. The problem of finding γ_r-function for the graph G is called the *weak Roman domination problem* (WRDP).

Although several structural results on the WRDP are known (see Henning and Hedetniemi (2003); Cockayne et al. (2004); Mai and Pushpam (2011); Chellali et al. (2014)), only few algorithmic results exist. For instance, integer linear programming (ILP) formulation, which corresponds to the Henning and Hedetniemi (2003) definition of the WRDP, was given in Ivanović (2017). Proof that the WRDP is NP-hard even when restricted to bipartite or chordal graphs is given in Henning and Hedetniemi (2003). Since the trivial enumeration algorithm for solving this problem runs in $O^*(3^n)$ in polynomial space, Chapelle et al. (2017) proposed algorithm for solving WRDP in $O^*(2^n)$ time needed exponential space, and $O^*(2.2279^n)$ algorithm using polynomial space (the notation $O^*(f(n))$ suppresses polynomial factors). In the same paper, it was shown that the WRDP can be solved in a linear-time on interval graphs. Linear-time algorithm for finding a WRDF for a block graph is proposed by Liu et al. (2010) but, as it will be shown in this paper, the proposed algorithm finds WRDF which is not always γ_r-function. Inspired by their algorithm, we developed an algorithm for solving the WRDP on a block graph which also runs in a linear-time.

Our paper is organized as follows. In Section 2 we give definitions and the algorithm for solving the WRDP on a block graph (known from the literature). Also, we give an example to illustrate that this algorithm does not find any γ_r-function for some block graphs. In Section 3, we present our newly developed algorithm for solving the WRDP on a block graph. Conclusion and References are given in the final two sections.

2. PRELIMINARIES AND NOTATIONS

Let $G = (V,E)$ be a graph with a vertex set V and a set of edges E. The following graph terms are taken from Harray (1969).

- A *subgraph* of G is a graph having all its vertices and edges in G. For any set S of vertices of G, a *induced subgraph* is the maximal subgraph of G with vertex set S (i.e. $H = (S,E')$, $S \subset V$ and for all vertices $u,v \in S$, $e = (u,v) \in E'$ iff $e \in E$).
- A *clique* is a subset of vertices of an undirected graph such that every two distinct vertices in that subset are adjacent, i.e. clique is an induced subgraph, which is complete.
- A graph G is *connected* if there is a path between every pair of its vertices.
  A graph G is *disconnected* if in G there are two vertices such that they are not endpoints of any path in G.
- A maximal connected subset $H = (V_H,E_H)$ of a graph G is called a *component of a graph* G if:
  1) for every two vertices $v_1,v_2 \in V_H$ $(v_1,v_2) \in E_H$ iff $(v_1,v_2) \in E$ and
  2) there is no path whose endpoints are in V_H and $V \setminus V_H$.
- A *cut-vertex* (articulation point) of a graph is the one whose removal increases the number of components.
- A *biconnected graph* is a connected and *non-separable graph*, meaning that the graph will remain connected after the removal of any vertex.
- A *biconnected component* (also known as a *block* or 2-connected component) is a maximal non-separable subgraph.
- If G is non-separable, then G itself is often called a *block*.
- A *bipartite graph* (or bi-graph) is a graph whose vertices can be divided into two disjoint and independent sets, U and V, such that every edge connects one vertex in U with one vertex in V.
- A tree T is a connected graph with no cycles.
- An undirected graph G is called an *intersection graph* if it is formed from a family of sets S_i, $i = 0,1,2,\ldots$ by creating one vertex v_i for each set S_i, and connecting two vertices v_i and v_j by an edge whenever the corresponding two sets have a nonempty intersection, that is $E(G) = \{\{v_i,v_j\}|S_i \cap S_j \neq \emptyset\}$.

For a rooted tree, we will use the same definition as it was used in Liu et al. (2010):

- The distance between two vertices u and v, denoted as $d(u,v)$, will be calculated as the minimum number of edges that we pass when walking from u to v; the distance between two adjacent vertices is equal to 1.
- A tree T is a rooted tree if it has a vertex r, called the *root* of the tree, vertices with degree 1, called a *leaves*, and non-leaf vertices, called *internal-nodes*.
- For two adjacent vertices u and v of the rooted tree T, vertex u will be called *parent* of v if $d(r,u) < d(r,v)$ with respect to the given root r. Vertex v will be called a *child of the vertex u*.
- Any vertex can have only one parent but many children.

Now, a *block graph* or a clique tree is a type of an undirected graph in which every biconnected component is a clique. Block graphs, derived from graph G, are usually denoted by $B(G)$, and can be defined as follows: blocks of G correspond to the vertices of $B(G)$ such that two vertices in $B(G)$ are adjacent whenever the corresponding blocks contain a common cut-vertex in G. Block graphs may be characterized as intersection graphs of blocks of arbitrary undirected graphs. For more details see Harary (1963).

For a connected block graph G, a *block-cut-vertex tree* (cut-tree for short) of G, denoted by $bc(G)$, is defined such that each node in the tree represents either a biconnected component or a cut-vertex of G, and the node that represents a cut-vertex is connected to all nodes representing any biconnected component which contains that cut-vertex.

A block graph and its cut-tree are illustrated in Figures 1 and 2.

In Figure 2, leaves B_1 and B_2 represent two component blocks, and vertices C_j, $j = 1,\ldots,5$ represent one component block (i.e. C_j is a block which contains only a cut-vertex of a graph G), while inner vertices B_i, $i = 3,\ldots,6$, represent empty blocks (since the corresponding cut-vertices of the graph G are adjacent and have an empty intersection, the intersection of the corresponding C_j vertices of the tree T_G will be empty sets, i.e. empty blocks).

In Liu et al. (2010), a function $f' : V \to \{0^-,0^+,1,2\}$ was designed such that the sign 0^- were used when a vertex was not dominated, and 0^+ if a vertex had been dominated at that moment. They also expected that only cut vertices could be of positive weight. Starting from a cut-tree T_G of a block graph G, they assigned 0^- to every leaf. Then, they computed weights of the inner vertices as follows. Let u be the current vertex.

If u is a cut vertex,

i. $f(u) = 2$ if there are more than two undefended vertices (vertices whose weights are equal to zero and whose examined neighbors are also with weights equal to zero) in the sets of children and grandchildren of u.

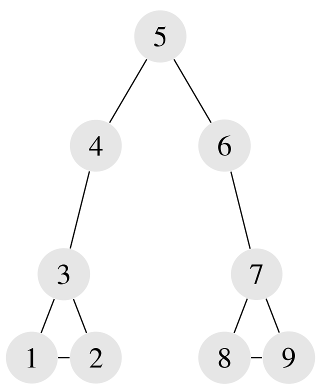
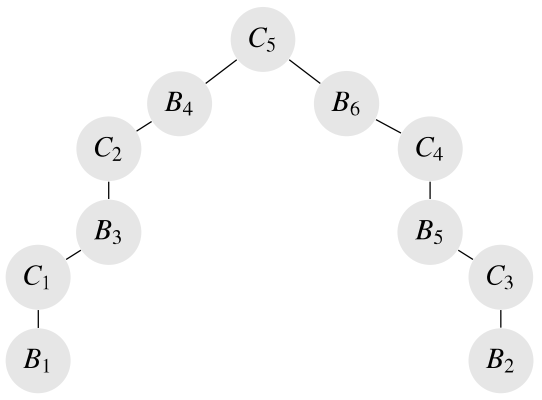


Figure 1 Block graph G



$B_1 = \{1,2\}$
$B_2 = \{8,9\}$
$B_3 = \emptyset$
$B_4 = \emptyset$
$B_5 = \emptyset$
$B_6 = \emptyset$
$C_1 = \{3\}$
$C_2 = \{4\}$
$C_3 = \{7\}$
$C_4 = \{6\}$
$C_5 = \{5\}$

Figure 2 Cut-tree of a block graph G, T_G

ii. $f(u) = 1$ if there is only one undefended vertex in the sets of children and grandchildren of u.
iii. $f(u) = 0^-$ if all children and grandchildren of u are either 0^+ or 1.
iv. $f(u) = 0^+$ if in the sets of children and grandchildren of u, there is a vertex v with $f(v) = 2$.

else

i. $f(u) = 0^+$ if the number of nodes in u is 0.
ii. $f(u) = 0^+$ if one children of u has value 2.
iii. $f(u) = 0^-$ otherwise.

Although Liu et al. (2010) claim that their algorithm finds a weak Roman domination function f of minimal weight, the found WRDF is not always γ_r-function. According to their algorithm, the minimal total weight of the cut-tree presented in Figure 2 is equal to 4: weight 2 is assigned to C_5, weight 1 to C_1 and C_3, and 0 to all other vertices. But, the optimal solution value of the illustrated cut-tree is equal to 3: weight 1 is assigned to C_1, C_3, and C_5, and weight equal to 0 to all other vertices. Indeed, if we apply the found solution to the corresponding block graph, we will notice that with weight of vertices 3, 5 and 7, which is equal to 1, and weight of all other vertices, which is equal to 0, all vertices will be considered as the currently defended and defended in case of one attack. More precisely, with the presented legion arrangement, in case of one attack, vertices 1 and 2 will be defended by the legion stationed at vertex 3; vertices 4 and 6 will be defended by the legion stationed at vertex 5; vertices 8 and 9 will be defended by the legion stationed at vertex 7, and all other vertices will be defended by their own stationed legion. So, all listed moves will not affect safety of any vertex.

3. LINEAR-TIME ALGORITHM FOR SOLVING THE WEAK ROMAN DOMINATION PROBLEM ON A BLOCK GRAPH

Based on the above considerations, we assume that G is a connected block graph. Its cut tree $T_G = (V_{T_G}, E_{T_G})$, which can be formed by depth first search in $O(|V| + |E|)$ time, is used as the input of our algorithm.

For a WRDF f let $f(i) \in \{0,1,2\}$ represents a number of legions assigned to a vertex i (let $f(i)$ be a weight of the vertex i). The output of the algorithm is γ_r-function, both for the cut-tree T_G and the block graph G, i.e. output is a function f, which assigns a value $f(i)$ to every vertex of the formed cut-tree T_G (i.e. for every vertex $i \in V_{T_G}$ that represents some cut-vertex of G, the corresponding cut-vertex in G will be of the same weight, while to all other vertices in G, which are not cut-vertices, the assigned weight is equal to 0) and is of minimal total weight for that tree.

Weight will be assigned to vertices of the cut-tree T_G post-order traversal (complexity of post-order traversal is $O(|V_{T_G}|)$), i.e. weight will be assigned first to leaves and last to the root of the tree; the weight can be assigned to a parent vertex only if it is assigned to all of its children vertices.

Since the WRDP is a dynamical problem, we say that a legion could be sent from one vertex (whose weight is positive) to its attacked neighbor vertex (whose weight is equal to zero); we decrease weight of the first vertex by one and increase weight of the second vertex by one. So, a vertex i can be considered as defended currently (i.e. it has positive weight or a neighbor j whose weight is positive), but it could be considered as undefended if its weight is equal to zero and the only neighbor j, whose weight is equal to one, sends a legion to defend neighbor k (i,k are neighbors to j, $k \neq i$), whose weight is also equal to zero. Therefore, we mark a vertex i as "defended by the stationed legion", "defended by the neighbors' legion", or "undefended", and we consider variants of these marks in accordance with the marks given for children and the parent. In other words, the mark of every vertex i can be determined by the flowchart-a shown in Figure 3.

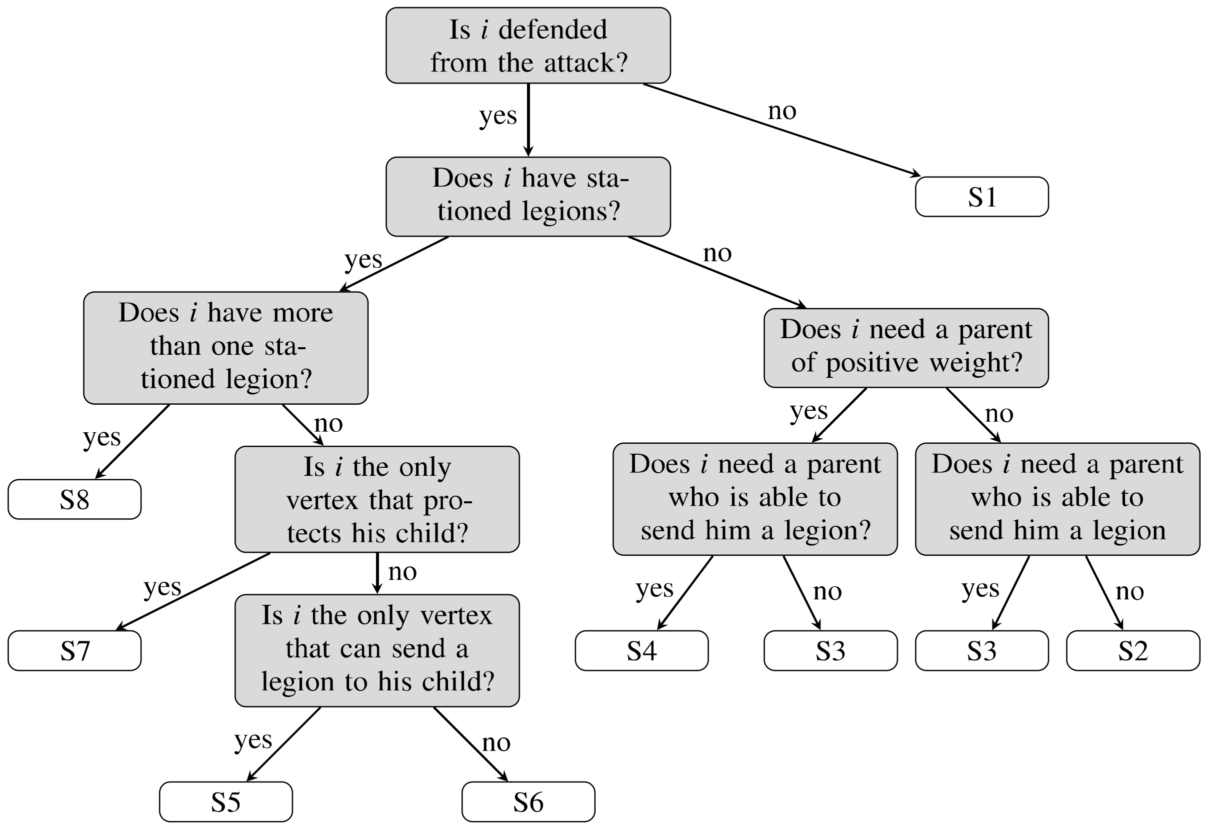

Is i defended from the attack?
yes
no
S1
Does i have stationed legions?
yes
no
Does i have more than one stationed legion?
Does i need a parent of positive weight?
yes
no
S8
Is i the only vertex that protects his child?
yes
no
Does i need a parent who is able to send him a legion?
Does i need a parent who is able to send him a legion
yes
no
S7
Is i the only vertex that can send a legion to his child?
S4
S3
S3
S2
yes
no
S5
S6

Figure 3 Flowchart of possible vertex marks

From the above, there are 8 different marks for every vertex i which can be interpreted as follows:

- S1 means that a vertex i has no child or his children don't have stationed legions, i.e. vertex i must have the parent who can protect him and send a legion in case of the attack; the parent can not send a legion to any other child or to the parent unless he has two stationed legions;
- S2 means that a vertex i has a child of positive weight j who can send a legion in case that it is attacked and, in case that a child k of j is attacked, j is not obligated to send a legion to k;
- S3 means that a vertex i has a child of positive weight j who can send a legion in case that it is attacked, and who also protects his own child k to whom it is obligated to send a legion when it is attacked. Therefore, a vertex i needs the parent with stationed legion to protect him when j sends his legion to the attacked child k. Also, the parent is not obligated to send him a legion in case when it is attacked;
- S4 means that a vertex i has a child of positive weight j and that j is the only neighbor with positive weight to his own child k, to whom he is obligated to send a legion when it is attacked. Therefore, a vertex i needs the parent who will protect him and is obligated to send a legion in case the child is attacked;
- S5 means that a vertex i has a stationed legion and does not have any unprotected child and is not obligated to send a legion to the children. Vertex i can send a legion to a parent in case that the parent is attacked;
- S6 means that a vertex i has one stationed legion and an unprotected child to whom it is not obligated to send a legion in case of the attack. Vertex i can send a legion to parent in case that parent is attacked;
- S7 means that a vertex i has one stationed legion and an unprotected child to whom it is obligated to send the legion in case that the child is attacked. Vertex i can not send the legion to its parent in case that the parent is attacked;
- S8 means that a vertex i has two stationed legions and, if it has an unprotected child, it can send one legion to attacked child, also it can send a legion to the parent in case that the parent is without station legions and needs a protection.

Our algorithm finds γ_r-function for the formed cut-tree T_G and the found solution is applied to the original block graph. Let $F(i)$ be equal to a number of legions stationed at vertex i, i.e. let $F(i)$ represents a weight of the vertex i. Since the weights of all vertices of the cut-tree T_G will be assigned post-order transversal, let i_1 represents the first and i_n, $n = |V_{T_G}|$ the last vertex to whom we assign their weights. Weight of a vertex i_m, $m = 1, \dots, n$, will be calculated as follows:

- If i_m represents a leaf, i_m will be marked with S1 and $F(i_m) = 0$.

- If i_m represents an empty block, it will be marked with S2 and $F(i_m) = 0$.
- Let i_m represents a nonempty block or a cut-vertex. If i_m represents a nonempty block, its mark will be determined in accordance with the marks of its children; in case that i_m represents a cut-vertex, its mark will be in accordance with the marks of its children and, whenever a child represents an empty-blocks, in accordance with the marks of the children of that child, too. For a vertex i_m, a number of the considered vertices with mark S1 will be denoted as num_1, a number of the considered vertices with mark S2 will be denoted as num_2, and so on (num_k represents a number of vertices with mark Sk of the considered vertices for a vertex i_m, $k = 1,\dots,8$).
  Depending on values num_k, $k = 1,\dots,8$, mark of the vertex i_m will be determined in the following way:

- If i_m is a non-empty block then, whenever conditions $num_2 + num_3 + num_4 = 0$ and $num_5 + num_6 + num_7 + num_8 > 0$ hold, it will be marked as
  1. S3 when $num_7 > 0$ and $num_5 + num_6 + num_8 = 0$,
  2. S2 when $num_5 + num_6 + num_8 > 0$,

  otherwise, it will be marked as S1. For all cases $F(i_m) = 0$.
- If i_m is a cut-vertex and
  1. one of the following two conditions hold:
     1) $num_1 > 0$ and $num_4 > 0$,
     2) $num_1 \geq 2$,

     it will be marked as S8, $F(i_m) = 2$.
  2. otherwise, if
     1) conditions $num_1 = 1$ and $num_4 = 0$ hold, it will be marked as S7.
     2) conditions $num_1 = 0$ and $num_4 > 0$ hold, it will be marked as S6.
     3) conditions $num_1 = 0$, $num_4 = 0$ and $num_3 > 0$ hold, it will be marked as S5.
     4) condition $num_1 + num_3 + num_4 = 0$ together with one of the following two conditions hold:
        i) $num_5 + num_8 > 0$,
        ii) $num_6 + num7 > 1$ i $num_6 > 0$,

     it will be marked as S2.
     5) conditions $num_1 + num_3 + num_4 + num_5 + num_6 + num_8 = 0$ and $num_7 > 0$ hold, it will be marked as S4.
     6) conditions $num_1 + num_3 + num_4 + num_5 + num_7 + num_8 = 0$ and $num_6 = 1$ hold, it will be marked as S3.
     7) conditions $num_1 + num_3 + num_4 + num_5 + num_6 + num_7 + num_8 = 0$ hold, it will be marked as S1.

  In cases 2.1)-2.3) $F(i_m) = 1$, while in cases 2.4)-2.7) $F(i_m) = 0$.

Since $F(i_m) \in \{0,1,2\}$ for every vertex i_m of the cut-tree T_G, and weights are computed so that every vertex has the positive weight, or has a neighbor whose weight is positive, with addition that, in case of an attack, it is possible to send a legion from one vertex with positive weight to the attacked neighbor without violating the safety of any vertex, constructed function F is the WRDF for the cut-tree T_G. Now, we will show that F is also the WRDF of minimal total weight.

Suppose the opposite, let F' be the WRDF for the cut-tree T_G of smaller total weight than the total weight of F. It is enough to show that total weight of F' is by one smaller than total weight of F. Therefore, let $F'(i) = F(i) - 1$ for some vertex i and $F'(j) = F(j)$ for all other vertices of the cut-tree T_G. Since $F'(i) \in \{0,1,2\}$, only two cases are possible for the vertex i: 1) $F(i) = 2$, and 2) $F(i) = 1$.

Let us first consider case 1). By the construction, $F(i) = 2$ holds if vertex i has at least two undefended children to whom it is the only adjacent vertex with positive weight and the only neighbor who can send a legion in case that someone attacks them. By the assumption, $F'(i) = F(i) - 1 = 1$ and $F'(j) = F(j)$ for all other vertices of the cut-tree T_G. Now, if a child of i, to whom i is obligated to send a legion in case of an attack, is attacked, i must send a legion in order to defend it. At this movement i will be left out of any stationed legion (will have weight equal to 0), and therefore other children, which were protected by the legion stationed at i, will become undefended, meaning that function F', constructed by the given assumption, is not a WRDF for the cut-tree T_G. So, this case is not possible.

Let us now consider case 2). By the construction of the function F, it follows that $F(i) = 1$ when

i) i has only one undefended child to whom it is the only neighbor with positive weight and to whom it is obligated to send a legion in case of the attack, and does not have any other undefended child or a grandchild (if child is an empty set) and is not obligated to send a legion to any of them.

ii) i has only one child (a grandchild if child is an empty set) to whom it is obligated to send a legion in case that the child is attacked and has no undefended children or grandchildren (if child is an empty set),

iii) i has no undefended children (nor grandchildren if child is an empty set), but has at least one child (grandchild if child is empty set) who needs the parent of positive weight.

Now, by the assumption, $F'(i) = F(i) - 1 = 0$ and $F'(j) = F(j)$ for all other vertices. In case i) i must be able to protect its unprotected child and must be able to send him a legion. Since $F'(i) = 0$, it follows that F', constructed by the assumption, is not a WRDF for the cut-tree T_G, the given child of i is left to be unprotected. In case ii) i must be able to send a legion to its child and with $F'(i) = 0$ we can conclude again that F', constructed by the given assumption, is not a WRDF for the cut-tree T_G. In case iii) weight of i is equal to zero, which is contrary to a child' needs, meaning again that F' can not be a WRDF for the cut-tree T_G.

So, since it not possible to lower the weight by one of any vertex whose weight is positive, it follows that total weight can not be lowered as well. Therefore, constructed WRDF F is with minimal total weight, implying that γ_r for the given T_G can be calculated as $\sum_{i \in V_{T_G}} F(i)$.

Now, since the weights of all vertices of the cut-tree T_G are known, constructed solution can be applied to the original block graph as follows: For every cut-vertex of the cut-tree T_G, the corresponding cut-vertex of the block graph will have the same weight. All other vertices will have weight equal to zero. Therefore, γ_r for the block graph can be also calculated as $\sum_{i \in V_{T_G}} F(i)$.

As it was earlier mentioned, the presented algorithm runs in a linear-time: transformation of the block graph into a cut-tree is in $O(|V| + |E|)$, vertices of the cut-tree are sorted in accordance with postfix index in $O(|V_{T_G}|)$, and the algorithm runs through every instance of the cut-tree T_G once, i.e. total minimal weight of that cut-tree is found in $O(|V_{T_G}|)$.

3.1. COMPUTATIONAL RESULTS

Results presented in Liu et al. (2010) are theoretical and without a source code or an executable file which can be used for testings. Therefore, in order to compare their algorithm with ours, we have programmed their algorithm and our newly developed algorithm in MATLAB2016b, and tested on Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz, with 8GB RAM under Windows 10. Testings were performed on the set of instances that can be downloaded from http://pallini.di.uniroma1.it/Graphs.html. Results of testings are presented in Table 1: Name, number of vertices and number of edges for every instance are given in the first three columns. The optimal solution value, calculated using optimization solver CPLEX and ILP formulation provided by Ivanović (2017), are given in the fourth column. The following two columns (*val* and *t*) contain value and execution time obtained by the algorithm developed by Liu et al. (2010). For every tested instance, the last two columns contain value (*val*) and execution time (*t*) obtained by the algorithm presented in this paper.

Table 1: Computational results

Instance				Alg. developed by Liu et al. (2010)		Our algorithm	
Name	|V|	|E|	*opt*	*val*	*t*	*val*	*t*
rantree-10	10	9	5	5	0.1037	5	0.0103
rantree-20	20	19	10	10	0.1511	10	0.0041
rantree-50	50	49	25	25	0.3022	25	0.0062
rantree-100	100	99	48	49	0.4254	48	0.0084
rantree-200	200	199	103	105	0.5338	103	0.0132
rantree-300	300	299	145	153	0.7563	145	0.0184
rantree-400	400	399	201	208	1.0028	201	0.0255
rantree-500	500	499	249	255	1.2237	249	0.0296
rantree-600	600	599	299	313	1.461	299	0.0381
rantree-700	700	699	346	360	1.7541	346	0.0401
rantree-800	800	799	391	403	2.5813	391	0.0469
rantree-900	900	899	452	463	2.1692	452	0.0541
rantree-1000	1000	999	490	510	2.4234	490	0.0590

Once more, as it can be seen from Table 1: our algorithm finds solution value equal to the optimal value (from column *opt*) for every tested instance, while the algorithm developed by Liu et al. (2010) finds solution value equal to the optimal value (from column *opt*) only for the first three tested instances and greater solution value for all other tested instances.

4. CONCLUSION

The algorithm for solving the weak Roman domination problem on a block graph, which is known from the literature, is considered in this paper. It is shown that the considered algorithm does not always compute the optimal solution value on a block graph and, therefore, a new linear-time algorithm for solving the weak Roman domination problem on a block graph is presented. Since the WRDP is NP-hard problem, we believe that our algorithm, with some modifications, can also be used for solving the WRDP on some other graph classes. Also, we think that a similar approach can be used for solving the Roman domination problem on a block graph as well.

Acknowledgement

This research was partially supported by Serbian Ministry of Education, Science and Technological Developments under grants no. TR36006 and ON174010.

REFERENCES

Ahangar, H., Henning, M., Löwenstein, C., Zhao, Y., and Samodivkin, V. (2014). Signed roman domination in graphs. *Journal of Combinatorial Optimization*, 27(2):241–255.

Beeler, R. A., Haynes, T. W., and Hedetniemi, S. T. (2016). Double roman domination. *Discrete Applied Mathematics*, 211:23–29.

Chambers, E., Kinnersley, B., Prince, N., and West, D. (2009). Extremal problems for roman domination. *SIAM Journal on Discrete Mathematics*, 23(3):1575–1586.

Chapelle, M., Cochefert, M., Couturier, J.-F., Kratsch, D., Letourneur, R., Liedloff, M., and Perez, A. (2017). Exact algorithms for weak roman domination. *Discrete Applied Mathematics*, in press.

Chellali, M., Haynes, T., and Hedetniemi, S. (2014). Bounds on weak roman and 2-rainbow domination numbers. *Discrete Applied Mathematics*, 178:27–32.

Cockayne, E., P.A.Jr., D., Hedetniemi, S., and Hedetniemi, S. (2004). Roman domination in graphs. *Discrete Mathematics*, 278 (1-3):11–22.

Dreyer, J. (2000). *Applications and variations of domination in graphs*. PhD thesis.

Harary, F. (1963). A characterization of block-graphs. *Canadian Mathematical Bulletin*, 6(1):1–6.

Harray, F. (1969). *GRAPH THEORY*. Addison-Wesley MR0256911.

Henning, M. (2002). A characterization of roman trees. *Discussiones Mathematicae Graph Theory*, 22(2):325–334.

Henning, M. and Hedetniemi, S. (2003). Defending the roman empire - a new strategy. *Discrete Mathematics*, 266:239–251.

Ivanović, M. (2017). Improved integer linear programming formulation for weak roman domination problem. *Soft Computing*, pages 1–11.

Liu, C. H. and Chang, G. J. (2013). Roman domination on strongly chordal graphs. *Journal of Combinatorial Optimization*, 26(3):608–619.

Liu, C. S., Peng, S. L., and Tang, C. Y. (2010). Weak roman domination on block graphs. In *The 27th Workshop on Combinatorial Mathematics and Computation Theory*, pages 86–89. Providence University, Taichung, Taiwan.

Mai, T. and Pushpam, P. (2011). Weak roman domination in graphs. *Discuss.Math.Graph Theory*, 31(1):161–170.

ReVelle, C. and Rosing, K. (2000). Defendens imperium romanum: a classical problem in military strategy. *Amer.Math.*, 107 (7):585–594.

Stewart, I. (1999). Defending the roman empire! *Sci. Amer.*, 281 (6):136–139.

Република Србија

Математички факултет Универзитета у Београду

ДИПЛОМА

о стеченом високом образовању

МАРИЈА Прославка ИВАНОВИЋ

рођена 06.11.1982. године у
Пожаревцу, Република Србија
уписана 2001/02. године, а дана 21.04.2007. године
завршила је студије на
Математичком факултету Универзитета у Београду
на смеру
нумеричка математика и оптимизација
са општим успехом 8.37 (осам и 37/100)
На основу тога јој се издаје ова
диплома о стеченом образовању
и стручном називу

Дипломирани математичар

Редни број из евиденције о издатим дипломама 24907.
У Београду 04.05.2007. године

В.Д. Декан
[illegible]
доцент др Драгољуб Кечкић

Ректор
[illegible]
проф. др Бранко Ковачевић

Република Србија

УБ

Универзитет у Београду
Математички факултет, Београд

Оснивач: Република Србија
Дозволу за рад број 612-00-02666/2010-04 од 10. децембра 2010. године је издало Министарство просвете и науке Републике Србије

Диплома

Марија, Радиша, Ивановић

рођена 6. новембра 1982. године у Пожаревцу, Република Србија, уписана школске 2011/2012. године, а дана 19. септембра 2012. године завршила је мастер академске студије, другог степена, на студијском програму Математика, обима 60 (шездесет) бодова ЕСПБ са просечном оценом 9,00 (девет и 0/100).

На основу тога издаје јој се ова диплома о стеченом високом образовању и академском називу

мастер математичар

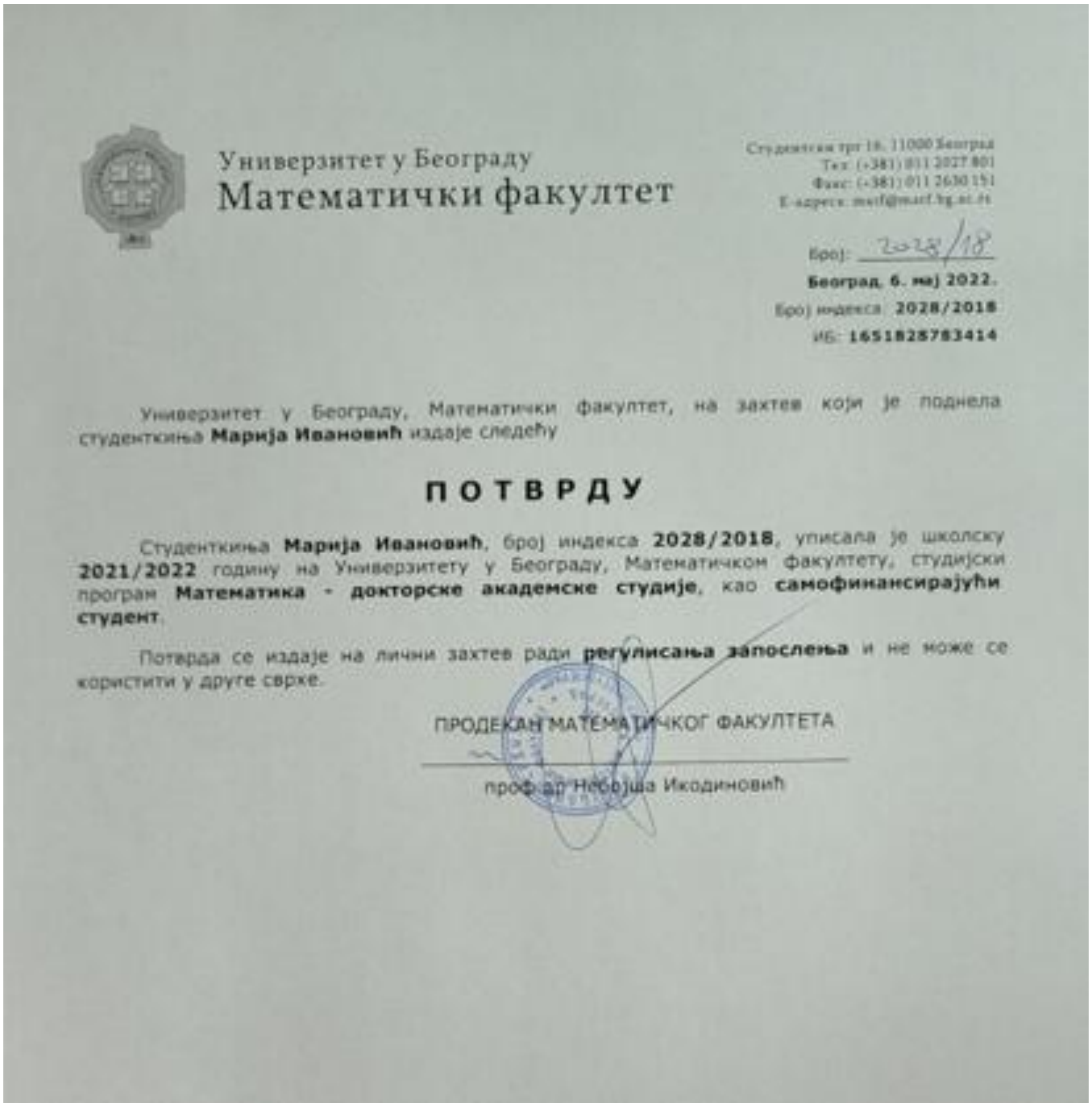
Број: 3699600

У Београду, 23. априла 2015. године

Декан
Проф. др Зоран Ракић
[illegible]

Ректор
Проф. др Владимир Бумбаширевић
[illegible]

00037143



Универзитет у Београду
Математички факултет

Студентски трг 16, 11000 Београд
Тел: (+381) 011 2027 801
Факс: (+381) 011 2630 151
E-адреса: matf@matf.bg.ac.rs

Број: 2028/18

Београд, 6. мај 2022.

Број индекса: **2028/2018**

ИБ: **1651828783414**

Универзитет у Београду, Математички факултет, на захтев који је поднела студенткиња **Марија Ивановић** издаје следећу

ПОТВРДУ

Студенткиња **Марија Ивановић**, број индекса **2028/2018**, уписала је школску **2021/2022** годину на Универзитету у Београду, Математичком факултету, студијски програм **Математика - докторске академске студије**, као **самофинансирајући студент**.

Потврда се издаје на лични захтев ради **регулисања запослења** и не може се користити у друге сврхе.

ПРОДЕКАН МАТЕМАТИЧКОГ ФАКУЛТЕТА

[illegible]

проф. др Небојша Икодиновић

УНИВЕРЗИТЕТ У БЕОГРАДУ

Адреса: Студентски трг 1, 11000 Београд, Република Србија
Тел.: 011 3207400; Факс: 011 2638818; E-mail: officebu@rect.bg.ac.rs

ВЕЋЕ НАУЧНИХ ОБЛАСТИ
ПРИРОДНО-МАТЕМАТИЧКИХ
НАУКА

Београд, 14.10.2019. године
02-04 Број 61206-3989/2-19
СЂ

На основу члана 48. став 5. тачка. 3. Статута Универзитета у Београду („Гласник Универзитета у Београду", број 201/18 и 207/19), и члана 32. Правилника о докторским студијама на Универзитету у Београду ("Гласник Универзитета у Београду", број 191/16), а на захтев Математичког факултета, број: 433/6 од 13.9.2019. године, Веће научних области природно-математичких наука, на седници одржаној 14.10.2019. године, донело је

О Д Л У К У

ДАЈЕ СЕ САГЛАСНОСТ на предлог теме докторске дисертације МАРИЈЕ ИВАНОВИЋ, под називом: „Нови приступи у решавању оптимизационог проблема римске доминације на графовима", и на одређивање проф.др Александра Савића за ментора.

ПРЕДСЕДНИК ВЕЋА

[illegible]

Проф. др Воја Радовановић

Доставити:
- Факултету
- архиви Универзитета