An improved Branch-and-cut code for the maximum balanced subgraph of a signed graph

Rosa Figueiredo, Yuri Y. Frota

An improved Branch-and-cut code for the maximum balanced subgraph of a signed graph

Rosa Figueiredo*
CIDMA, Department of Mathematics, University of Aveiro
3810-193 Aveiro, Portugal.

Yuri Frota
Department of Computer Science, Fluminense Federal University
24210-240 Niterói-RJ, Brazil.

Abstract

The Maximum Balanced Subgraph Problem is the problem of finding a subgraph of a signed graph that is balanced and maximizes the cardinality of its vertex set. We are interested in the exact solution of the problem: an improved version of a branch-and-cut algorithm is proposed. Extensive computational experiments are carried out on a set of instances from three applications previously discussed in the literature as well as on a set of random instances.

Keywords: Balanced signed graph; Branch-and-cut; Portfolio analysis; Network matrix; Community structure.

1 Introduction

Let $G = (V, E)$ be an undirected graph where $V = \{1, 2, \ldots, n\}$ is the set of vertices and E is the set of edges connecting pairs of vertices. Consider a function $s : E \to \{+, -\}$ that assigns a sign to each edge in E. An undirected graph G together with a function s is called a *signed graph*. An edge $e \in E$ is called *negative* if $s(e) = -$ and *positive* if $s(e) = +$.

In the last decades, signed graphs have shown to be a very attractive discrete structure for social network researchers [1, 8, 9, 16, 21] and for researchers in other scientific areas, including portfolio analysis in risk management [14, 15], biological systems [7, 15], efficient document classification [3], detection of embedded matrix structures [12] and community structure [17, 20]. The common element among all these applications is that all of them are defined in a collaborative vs. conflicting environment represented over a signed graph. We refer the reader to [22] for a bibliography of signed graphs. In this work, we consider the Maximum balanced subgraph problem (MBSP) defined next.

Let $G = (V, E, s)$ denote a signed graph and let E^- and E^+ denote, respectively, the set of negative and positive edges in G. Also, for a vertex set $S \subseteq V$, let $E[S] = \{(i, j) \in E \mid i, j \in S\}$ denote the subset of edges induced by S. A signed graph $G = (V, E, s)$ is *balanced* if its vertex set can be partitioned into sets W (possibly empty) and $V \setminus W$ in such a way that $E[W] \cup E[V \setminus W] = E^+$. Given a signed graph $G = (V, E, s)$, the MBSP is the problem of finding a subgraph $H = (V', E', s)$ of G such that H is balanced and maximizes the cardinality of V'.

The MBSP is known to be an NP-hard problem [6] although the problem of detecting balance in signed graphs can be solved in polynomial time [13]. In the literature, the MBSP has already been applied in the detection of embedded matrix structures [10, 11, 12], in portfolio analysis in risk management [10] and community structure [10].

The problem of detecting a maximum embedded reflected network (DMERN) is reduced to the MBSP in [12]. Most of the existing solution approaches to the MBSP were in fact proposed for the solution of the DMERN problem. The literature proposes various heuristics for the solution of the DMERN problem (for references see [12]). Lately, Figueiredo et al. [11] developed the first exact solution approach for the MBSP: a branch-and-cut algorithm based on the signed graph reformulation from Gulpinar et al. [12] for the DMERN problem. Computational experiments were carried out over a set of instances found in the literature as a test set for the DMERN problem. Almost all these instances were solved to optimality in a few seconds showing that they were not appropriate for assessing the quality of a heuristic approach to the problem. Recently, Figueiredo et al. [10] introduced applications of the MBSP in other two different research areas: portfolio analysis in risk management and community structure. These authors also provided a new set of benchmark instances of the MBSP (including a set of difficult instances for the DMERN problem) and contributed to the efficient solution of the problem by developing a pre-processing routine, an efficient GRASP metaheuristic, and improved versions of a greedy heuristic proposed in [12].

In this work we contribute to the efficient solution

*Corresponding author. Departamento de Matemática, Campus de Santiago, Universidade de Aveiro, 3810-193 Aveiro, Portugal. Fax number: +351 234370066, Email: rosamaria.figueiredo@gmail.com

of the MBSP by developing an improved version of the branch-and-cut algorithm proposed by Figueiredo et al. [11]. We introduce a new branching rule to the problem based on the odd negative cycle inequalities. Moreover, we improve the cut generation component of the branch-and-cut algorithm by implementing new separation routines and by using a cut pool separation strategy.

The remainder of the paper is structured as follows. The integer programming formulation and the branch-and-cut algorithm proposed in [11] to the MBSP are outlined in Section 2. The improved version of the branch-and-cut algorithm is described in Section 3. In Section 4, computational results are reported for random instances as well as for instances of the three applications previously mentioned. In Section 5 we present concluding remarks.

We next give some notations and definitions to be used throughout the paper. For an edge set $B \subseteq E$, let $G[B]$ denote the subgraph of G induced by B. A set $K \subseteq V$ is called a *clique* if each pair of vertices in K is joined by an edge. A set $I \subseteq V$ is called a *stable set* if no pair of vertices in I is joined by an edge. We represent a cycle by its vertex set $C \subseteq V$. In this text, a signed graph is allowed to have parallel edges but no loops. Also, we assume that parallel edges have always opposite signs.

2 Integer programming formulation and branch-and-cut

The integer programming formulation and the branch-and-cut algorithm introduced in [11] are described next.

2.1 Integer programming formulation

It is well known that a signed graph is balanced if and only if it does not contain a parallel edge or a cycle with an odd number of negative edges [5, 12, 22]. Let $C^o(E)$ be the set of all odd negative cycles in G, i.e., cycles with no parallel edges and with an odd number of negative edges. Throughout this text, a cycle $C \in C^o(E)$ is called an *odd negative cycle*. The formulation uses binary decision variables $y \in \{0,1\}^{|V|}$ defined in the following way. For all $i \in V$, y_i is equal to 1 if vertex $i \in V$ belongs to the balanced subgraph, and is equal to 0 otherwise. We use the vector notation $y = (y_i)$, $i \in V$, and the notation $y(V') = \sum_{i \in V'} y_i$ for $V' \subseteq V$. The formulation follows.

$$\text{Maximize } y(V) \tag{1}$$

$$\text{subject to } y_i + y_j \leq 1, \quad \forall\ (i,j) \in E^- \cap E^+, \tag{2}$$

$$y(C) \leq |C| - 1, \quad \forall\ C \in C^o(E), \tag{3}$$

$$y_i \in \{0,1\}, \quad \forall\ i \in V. \tag{4}$$

Consider a parallel edge $(i,j) \in E^- \cap E^+$. Constraints (2) ensure vertices i and j cannot belong together to the balanced subgraph. Constraints (3), called *odd negative cycle inequalities*, forbid cycles with an odd number of negative edges in the subgraph described by variables y. These constraints force variables y to define a balanced subgraph. Finally, the objective function (1) looks for a maximum balanced subgraph. The formulation has n variables and, due to constraints (3), might have an exponential number of constraints. Let us refer to this formulation as $Y(G,s)$. By changing the integrality constraints (4) in formulation $Y(G,s)$ by the set of trivial inequalities $0 \leq y_i \leq 1$, $i \in V$, we obtain a linear relaxation to the MBSP.

2.2 A branch-and-cut algorithm

The branch-and-cut algorithm developed in [11] is based on formulation $Y(G,s)$, uses a standard 0–1 branching rule and has three basic components: the initial formulation, the cut generation and the primal heuristic.

Initial formulation The initial formulation is defined as

$$\text{maximize } y(V)$$

$$\text{subject to } y(K) \leq 1, \quad \forall\ K \in L, \tag{5}$$

$$y(C) \leq |C| - 1, \quad \forall\ C \in M \subseteq C^o(E), \tag{6}$$

$$y(K) \leq 2, \quad \forall\ K \in N, \tag{7}$$

$$0 \leq y_i \leq 1, \quad \forall\ i \in V, \tag{8}$$

where (5) are clique inequalities from the stable set problem [19] defined over a set of cliques L in $G[E^+ \cap E^-]$; (6) is a subset of inequalities (3) defined over a set of odd negative cycles M; (7) is a subset of inequalities from a family of negative clique inequalities introduced in [11] for the MBSP and defined over a set of cliques N in $G[E^-]$; (8) is the set of trivial inequalities. Greedy procedures described in [11] are used to generate sets L, M and N.

Cut generation After an LP has been solved in the branch-and-cut tree, the algorithm check if the solution is integer feasible. If this is not the case, the cut generation procedure is called and a set of separation routines is executed (a limit of 100 cuts per iteration is set). If no violated inequality is found or if a limit of 10 cut generations rounds is reached, the algorithm enter in the branching phase. The cut generation component described in [11] has two separation procedures. An exact separation procedure is used to generate violated odd negative cycle inequalities (3). This separation routine is based on a polynomial algorithm described in [4] to solve the separation problem for cut inequalities. A heuristic separation procedure defined in [11] is used to generate violated clique inequalities also introduced in [11].

Primal heuristic and branching rule A rounding primal heuristic is executed in [11] every time a fractional solution is found. Moreover, a standard 0–1 branching rule is used with the same branching priority assigned to each variable and the branch-and-cut tree is investigated with the best-bound-first strategy.

3 An improved branch-and-cut code

In this work, the following new routines were added to the branch-and-cut algorithm described in Section 2.

3.1 Branching on the odd negative cycle inequalities

Our branching rule is based on the odd negative cycle inequalities (3). The intuition behind this cycle based branching is the attempt to generate more balanced enumerative trees. The standard 0–1 branching rule can be very asymmetrical producing unbalanced enumerative trees.

Let $\bar{y} \in \mathbb{R}$ be the optimal fractional solution of a node in the search tree. Let $C' \subseteq C^o(E)$ be the subset of odd negative cycles such that each cycle $C \in C'$ satisfy the following conditions:

- constraint (3) defined by C' is a binding one in the current formulation,
- there exists a vertex $i \in C'$ such that $\bar{y}_i$ is fractional.

The standard 0–1 branching rule is used whenever C' is an empty set. If it is not the case, let $\bar{C}$ be the smallest cycle in C'. Split $\bar{C}$ into the sets $\bar{C}^1$ and $\bar{C}^2$ such that $\bar{C} = \bar{C}^1 \cup \bar{C}^2$, $\bar{C}^1 \cap \bar{C}^2 = \emptyset$ and $y(\bar{C}^1)$ is fractional. We create three branches in the search tree:

(i) $y(\bar{C}^1) \leq |\bar{C}^1| - 1$ and $y(\bar{C}^2) = |\bar{C}^2|$;

(ii) $y(\bar{C}^1) = |\bar{C}^1|$ and $y(\bar{C}^2) \leq |\bar{C}^2| - 1$;

(iii) $y(\bar{C}^1) \leq |\bar{C}^1| - 1$ and $y(\bar{C}^2) \leq |\bar{C}^2| - 1$.

3.2 Separation routines

In this work, we introduce two new separation procedures to the cut generation component of the branch-and-cut algorithm described in Section 2.

The authors in [11] proved that lifted odd hole inequalities (from the stable set problem) defined over the set of parallel edges $E^+ \cap E^-$ are valid inequalities for the MBSP. They have also proved that, if the support graph of these inequalities satisfy certain conditions they are facet defining inequalities to the problem. We implemented a separation procedure described in [18] to the lifted odd hole inequalities. Also, the authors indicated in [11] that a very similar lifting procedure could be applied to strengthen constraints (3). We implemented this lifting procedure to the odd negative cycle inequalities satisfying $|C| \leq 20$. In both cases, a very small instance of the MBSP must be solved at each iteration of the lifting procedures. In our implementation, these small problems were solved by simple enumerative algorithms.

Moreover, we added a cut pool to the branch-and-cut code: any violated inequality included to the active formulation of a node in the branch-and-cut tree is also included in the cut pool. As we have mentioned in Section 2, after an LP has been solved in the branch-and-cut tree, we check if the solution is integer feasible. If this is not the case, the cut generation procedure is then called. Before running any separation routine from our cut generation procedure, we check if there are violated cuts in the cut pool. In positive case, no separation routine is called and the violated cuts (limited to 100 cuts) are immediately added to the active formulation.

4 Computational experiments

We implemented and compared five versions of the branch-and-cut algorithm; each one of them uses the initial formulation defined by (5)-(8).

- BC: Branch-and-cut algorithm described in Section 2.
- BC+Cut: BC with the additional separation routines described in Section 3.2.
- BC+Cycle: BC with the cycle branching rule described in Section 3.1.
- BC+Cut+Cycle: BC with the additional procedures described in Sections 3.2 and 3.1.
- BC+Balas: BC with a branching rule proposed in [2] and successfully applied to solve the stable set problem.

Each version of the branch-and-cut algorithm was implemented in C++ running on a Intel(R) Pentium(R) 4 CPU 3.06 GHz, equipped with 3 GB of RAM. We use Xpress-Optimizer 20.00.21 to implement the components of these enumerative algorithms. The maximum running time per instance was set at 3600 seconds. The instance classes reported in [10] were tested here to allow for a better comparison of the performances of the branch-and-cut algorithms implemented. The class *Random* consists of 216 randomized instances divided into two groups: Group 1 without parallel edges and Group 2 with parallel edges. The class *UNGA* is composed of 63 instances derived from the community structure of networks representing voting on resolutions in the United Nations General Assembly. The class new *DMERN* consists of 316 signed graphs coming from a set of general mixed integer programs. Finally, the class *Portifolio* is composed by 850 instances generated from market graphs. The entire benchmark is available for download in `www.ic.uff.br/~yuri/mbsp.html`.

We first investigate the behavior of the *Random* instances, the results obtained by the five methods are summarized in Table 2. This table exhibits, for both groups, average times per $|V|$, and average percentage gaps per $|V|$, d (density of the graph) and the rates $|E^-| / |E^+|$ and $|E^+ \cap E^-|$. Multicolumn Time, gives us average times (in seconds) spent to solve instances to optimality; the values in brackets show the number of instances solved to optimality ("-" means no instance was solved within the time limit). Multicolumn %Gap presents the average of percentage gaps calculated over the set of unsolved instances. The percentage gap of each

instance is calculated between the best integer solution found and the final upper bound. For each group of instances, each line presents the results obtained with each one of the branch-and-cut algorithms implemented (in the same order they were described above). For Group 1, results obtained with version BC+Cut+Cycle are better. Compared with the original version BC, two more instances were solved to optimality, both percentage gaps and time were reduced. For Group 2, results obtained with version BC+Cut are slightly better. The version BC+Balas solved instances with 150 and 200 vertices not solved by any other version. However, this version got the worst results for instances in Group 1 and the highest gaps for unsolved instances in both groups.

In the second experiment, we analyze the performance of the algorithms on *Portifolio* instances. Table 3 reports the results obtained on instances with under 330 vertices. The first two columns give the number of vertices and a threshold value t used to generate the instances [10]. The next multicolumns give us, for each version implemented, averages for time, percentage gap (as defined in Table 2) and number of evaluated nodes in the search tree calculated for instances with a same value of $|V|$ and t. The last two lines reports the total number of instances solved to optimality, the average gap calculated for all instances and the maximum gap (in brackets). Version BC+Cut+Cycle solved more instances to optimality (227 out of 850 while the original BC managed to solve only 218 instances) and small gaps. The average (maximum) gaps for the original BC over the set of all unsolved instances is 22.12% (76.94%), while the same value for the improved version BC+Cut+Cycle is 12.27% (67.82%). Figure 1 reports gaps obtained with versions BC and BC+Cut+Cycle on each instance with under 270 vertices (instances with less than 270 vertices are easy instances solved to optimality in less than 1 minute). The x-axis exhibits the instances ordered primarily by number of vertices and secondly by the threshold value t. We can see that BC+Cut+Cycle presents tighter gaps for almost the entire set of *Portifolio* instances. We can also conclude these instances become more difficult as the number of vertices increases and the threshold value t decreases.

In the third experiment, we investigate the behavior of the *UNGA* instances. We notice that these instances are extremely easy to solve. No matter the number of vertices or the parameters used to compose the instance, all versions were able to solve all of them in a few seconds and in the root of the search tree. Hence, we could not draw any conclusion from this class of instance.

In our last experiment, all versions were applied to each one of the 316 new *DMERN* instances. Both, the number of vertices and edges, vary arbitrarily on the instances in this set (see [10]) which makes difficult to group them by similarity. Table 1 shows average results calculated over the 316 instances. The notations in this table are the same as in the previous ones. From this set of instances, we can extract 25 instances not solved to optimality by the original BC code with average (maximum) gap of 30.80% (274.67%) while version BC+Cut+Cycle could not solve 20 instances with a tighter average (maximum) gap of 16.22% (85.43%). One can also notice that the implementation of the new separation routines and the cycle branching rule used in BC+Cut+Cycle led to a better performance and a high number of evaluated nodes within the time limit.

Table 1: Results obtained on New DMERN instances.

	Time	%Gap(max)	Nodes
BC	69.85(291)	30.80(274.67)	91.33
BC+Cut	80.24(291)	31.20(318.39)	90.26
BC+Cycle	85.68(296)	16.21(85.43)	91.33
BC+Cut+Cycle	86.67(296)	16.22(85.43)	133.74
BC+Balas	70.06(289)	19.75(171.66)	521.78

5 Final remarks

In this work, we proposed an improved branch-and-cut algorithm based on the integer programming formulation and the BC algorithm proposed in [11], together with a new branching rule based on the odd negative cycle inequalities and improved cutting plane routines and strategies. The instance classes described in [10] were used to compare the performances of five variations of the branch-and-cut algorithm proposed in [11]. The results obtained by the version containing all improvements proposed in this work were superior to those given by the previously existing branch-and-cut. The improved version BC+Cut+Cycle solved 1100 out of 1445 instances within 1 hour of processing time, while the original algorithm managed to solve 1082 instances. Moreover, as we saw in Section 4, considering the set of unsolved instances, for most part of the cases, the average and maximum gaps obtained with this improved BC was smaller than the average and maximum gaps obtained with the original BC from [11]. The optimal values and bounds obtained by this improved version were used in [10] to evaluate the quality of heuristic results.

Acknowledgements

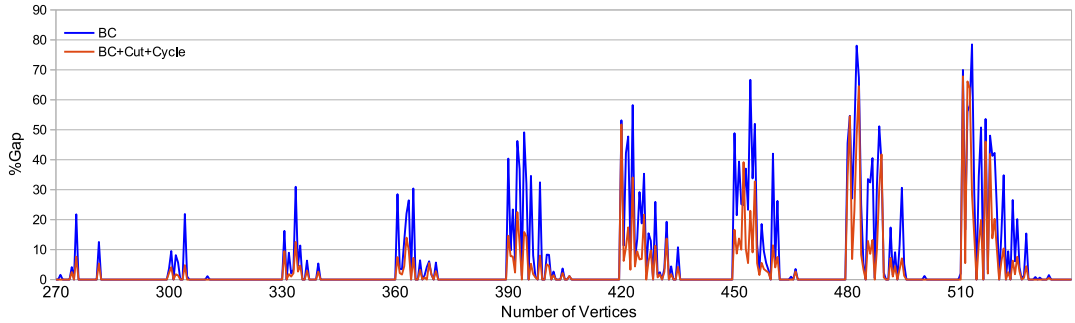
This work was supported by Portuguese funds through the CIDMA - Center for Research and Development in Mathematics and Applications, and the Portuguese Foundation for Science and Technology ("FCT-Fundação para a Ciência e a Tecnologia"), within project PEst-OE/MAT/UI4106/2014.

References

[1] P. Abell and M. Ludwig. Structural balance: a dynamic perspective. *Journal of Mathematical Sociology*, 33:129–155, 2009.

[2] E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 14:1054–1068, 1986.

Figure 1: Results obtained on portfolio instances.



[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *Proceedings of the 43rd annual IEEE symposium of foundations of computer science*, pages 238–250, Vancouver, Canada, 2002.

[4] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[5] F. Barahona and A.R. Mahjoub. Facets of the balanced (acyclic) induced subgraph polytope. *Mathematical Programming*, 45:21–33, 1989.

[6] J.J. Barthold. A good submatrix is hard to find. *Operations Research Letters*, 1:190–193, 1982.

[7] B. DasGupta, G. A. Encisob, E. Sontag, and Y. Zhanga. Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. *BioSystems*, 90:161–178, 2007.

[8] P. Doreian and A. Mrvar. A partitioning approach to structural balance. *Social Networks*, 18:149–168, 1996.

[9] P. Doreian and A. Mrvar. Partitioning signed social networks. *Social Networks*, 31:1–11, 2009.

[10] R. Figueiredo and Y. Frota. The maximum balanced subgraph of a signed graph: applications and solution approaches. *European Journal of Operational Research*, 236(2):473–487, 2014.

[11] R. Figueiredo, M. Labbé, and C.C. de Souza. An exact approach to the problem of extracting an embedded network matrix. *Computers & Operations Research*, 38:1483–1492, 2011.

[12] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovitch. Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137:359–372, 2004.

[13] F. Harary and J.A. Kabell. A simple algorithm to detect balance in signed graphs. *Mathematical Social Sciences*, 1:131–136, 1980.

[14] F. Harary, M. Lim, and D. C. Wunsch. Signed graphs for portfolio analysis in risk management. *IMA Journal of Management Mathematics*, 13:1–10, 2003.

[15] F. Huffner, N. Betzler, and R. Niedermeier. Separator-based data reduction for signed graph balancing. *Journal of Combinatorial Optimization*, 20:335–360, 2010.

[16] T. Inohara. On conditions for a meeting not to reach a deadlock. *Applied Mathematics and Computation*, 90:1–9, 1998.

[17] K.T. Macon, P.J. Mucha, and M.A. Porter. Community structure in the united nations general assembly. *Physica A: Statistical Mechanics and its Applications*, 391:343—-361, 2012.

[18] M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[19] S. Rebennack. *Encyclopedia of optimization*, chapter Stable set problem: branch & cut algorithms, pages 3676–3688. Springer, 2008.

[20] V.A. Traag and J. Bruggeman. Community detection in networks with positive and negative links. *Physical Review E*, 80:036115, 2009.

[21] B. Yang, W.K. Cheung, and J. Liu. Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering*, 19:1333–1348, 2007.

[22] T. Zaslavsky. A mathematical bibliography of signed and gain graphs and allied areas. *Electronic Journal of Combinatorics DS8*, 1998.

Table 2: Results obtained on random instances in Group 1 ($E^- \cap E^+ = \emptyset$) and in Group 2 ($E^- \cap E^+ \neq \emptyset$).

Instances	Time				%Gap													
	$\|V\|$				$\|V\|$				d			$\|E^-\|/\|E^+\|$			$(\|E^- \cap E^+\|)/\|E\|$		
	50	100	150	200	50	100	150	200	.25	.50	.75	.50	1	2	.25	.50	.75
Group 1	51.04(27)	2985.00(3)	–	–	0.00	47.76	106.01	157.08	80.80	91.68	84.75	77.79	90.70	86.85	–	–	–
	75.00(27)	2933.33(3)	–	–	0.00	47.33	105.36	156.77	80.38	91.30	84.23	76.55	91.09	86.50	–	–	–
	10.56(27)	1802.00(5)	–	–	0.00	28.07	94.52	151.72	69.09	83.86	76.23	70.26	79.62	75.61	–	–	–
	10.52(27)	1884.75(5)	–	–	0.00	28.46	93.19	146.15	66.92	82.56	76.26	66.68	79.31	75.44	–	–	–
	65.15(27)	2546.00(1)	–	–	0.00	63.02	119.25	179.91	88.22	110.29	107.38	89.61	107.07	101.13	–	–	–
Group 2	1.04(27)	411.20(20)	1827.25(8)	–	0.00	7.49	44.05	100.64	31.48	49.38	59.90	–	–	–	64.70	36.04	19.03
	1.22(27)	543.52(21)	1827.75(8)	–	0.00	7.87	42.94	99.60	31.35	48.70	59.26	–	–	–	64.60	34.80	19.05
	1.44(27)	830.96(24)	2095.67(6)	–	0.00	2.61	46.47	109.28	31.39	52.80	64.41	–	–	–	62.79	40.37	22.63
	1.59(27)	558.81(21)	2109.83(6)	–	0.00	4.14	46.51	108.69	31.95	52.65	64.42	–	–	–	63.59	40.18	22.58
	1.67(27)	344.80(20)	1281.67(9)	2979.67(3)	0.00	12.81	53.79	117.81	40.00	63.78	61.52	–	–	–	82.45	46.11	14.81

Table 3: Results obtained on portfolio instances.

Instance		BC			BC+Cut			BC+Cycle			BC+Cut+Cycle			BC+Balas		
$\|V\|$	t	Time	%Gap	#Nodes	Time	%Gap	#Nodes	Time	%Gap	#Nodes	Time	%Gap	#Nodes	Time	%Gap	#Nodes
330	0.300	910.00(3)	11.06	801.90	530.00(3)	5.25	792.40	30.50(2)	9.80	1012.60	226.00(3)	4.90	914.90	926.33(3)	11.06	800.80
	0.325	271.75(8)	5.84	434.30	127.38(8)	3.71	303.70	105.50(8)	4.14	454.10	85.50(8)	2.80	414.00	275.88(8)	5.82	443.40
	0.350	12.30(10)	–	14.80	24.60(10)	–	45.30	15.20(10)	–	19.00	21.00(10)	–	37.20	12.60(10)	–	14.80
	0.375	0.50(10)	–	1.30	0.50(10)	–	2.70	0.50(10)	–	2.00	0.50(10)	–	2.60	0.50(10)	–	1.30
	0.400	0.00(10)	–	1.00	0.00(10)	–	1.00	0.00(10)	–	1.00	0.00(10)	–	1.00	0.00(10)	–	1.00
360	0.300	1153.67(3)	18.21	544.00	216.00(3)	6.66	440.60	702.00(3)	17.09	558.00	197.00(3)	6.68	553.90	1156.33(3)	18.21	545.40
	0.325	151.25(4)	4.26	593.20	733.20(5)	2.68	741.50	348.25(4)	3.00	856.80	334.80(5)	2.79	849.30	156.00(4)	4.26	592.40
	0.350	163.70(10)	–	98.70	168.70(10)	–	139.10	262.20(10)	–	220.20	135.30(10)	–	134.90	160.00(10)	–	98.70
	0.375	1.90(10)	–	2.20	2.60(10)	–	4.00	2.40(10)	–	2.30	3.30(10)	–	6.20	1.90(10)	–	2.20
	0.400	0.10(10)	–	1.40	0.10(10)	–	1.30	0.10(10)	–	1.40	0.20(10)	–	1.50	0.10(10)	–	1.40
390	0.300	141.00(1)	27.21	502.90	1043.50(2)	12.13	462.60	78.00(1)	22.17	545.00	666.00(2)	12.15	432.40	142.00(1)	27.21	502.50
	0.325	261.00(4)	15.43	460.90	144.00(4)	5.13	399.40	151.50(4)	13.08	513.50	102.75(4)	4.24	499.30	256.75(4)	15.43	459.70
	0.350	74.14(7)	2.50	350.60	61.14(7)	1.52	384.60	39.00(7)	2.22	426.20	28.43(7)	1.61	556.20	72.86(7)	2.50	349.50
	0.375	3.00(10)	–	2.40	3.40(10)	–	3.60	4.50(10)	–	4.40	3.90(10)	–	3.90	3.00(10)	–	2.40
	0.400	0.10(10)	–	1.10	0.10(10)	–	1.10	0.20(10)	–	1.10	0.30(10)	–	1.70	0.10(10)	–	1.10
420	0.300	–(0)	29.05	396.80	–(0)	29.05	396.80	-(0)	26.23	413.20	–(0)	15.08	386.30	–(0)	29.13	395.60
	0.325	1062.00(2)	12.54	436.80	1062.00(2)	12.54	436.80	914.00(2)	12.84	499.30	1466.00(3)	7.68	536.60	1067.50(2)	12.54	435.90
	0.350	128.86(7)	11.49	246.80	128.86(7)	11.49	246.80	139.14(7)	8.84	323.30	117.29(7)	6.58	301.50	132.71(7)	11.49	246.60
	0.375	198.40(10)	–	131.90	198.40(10)	–	131.90	370.60(10)	–	348.20	146.90(10)	–	180.70	198.70(10)	–	131.90
	0.400	4.80(10)	–	11.40	4.80(10)	–	11.40	2.50(10)	–	4.80	5.40(10)	–	21.70	4.60(10)	–	11.40
450	0.300	-(0)	37.25	307.60	-(0)	16.25	298.20	-(0)	33.31	307.80	-(0)	16.87	312.30	-(0)	37.27	307.90
	0.325	352.00(1)	13.90	357.50	189.00(1)	7.02	362.70	495.00(1)	11.63	386.10	128.00(1)	4.92	354.90	343.00(1)	13.90	356.20
	0.350	397.00(8)	2.25	226.40	470.44(9)	2.65	218.20	468.56(9)	4.20	207.10	400.44(9)	2.68	253.30	396.50(8)	2.25	226.20
	0.375	17.00(10)	–	8.40	21.20(10)	–	17.20	17.70(10)	–	7.80	23.90(10)	–	19.00	17.10(10)	–	8.40
	0.400	1.30(10)	–	1.40	1.70(10)	–	2.80	1.50(10)	–	1.00	1.50(10)	–	1.00	1.30(10)	–	1.40
480	0.300	2083.00(1)	42.90	243.40	466.00(1)	33.25	237.90	1289.00(1)	38.95	244.40	747.00(1)	28.43	249.00	2086.00(1)	42.90	242.70
	0.325	800.50(2)	26.91	314.00	530.00(3)	18.12	276.50	560.00(2)	26.76	327.60	554.33(3)	16.03	285.20	799.50(2)	26.91	312.70
	0.350	364.20(5)	11.62	279.20	160.60(5)	3.95	256.50	212.40(5)	8.65	286.80	224.60(5)	3.45	310.40	362.80(5)	11.62	278.10
	0.375	29.67(9)	1.20	93.90	128.50(10)	–	51.30	59.44(9)	0.54	169.30	177.00(10)	–	77.60	29.67(9)	1.20	94.60
	0.400	18.40(10)	–	21.10	4.10(10)	–	4.90	8.70(10)	–	10.90	6.20(10)	–	7.00	18.80(10)	–	21.10
510	0.300	2812.00(1)	43.98	197.30	1710.50(2)	36.88	182.40	-(0)	36.43	210.20	950.00(2)	34.78	186.10	2813.00(1)	43.98	197.40
	0.325	446.50(2)	32.78	220.70	712.50(2)	15.62	243.10	333.00(2)	29.92	213.30	458.50(2)	18.83	242.20	442.00(2)	32.78	221.10
	0.350	46.33(3)	11.90	237.20	49.33(3)	3.74	244.20	48.00(3)	10.44	252.00	57.67(3)	3.70	305.00	46.33(3)	11.90	237.30
	0.375	95.29(7)	0.99	273.40	53.14(7)	0.60	375.70	412.25(8)	1.14	349.90	708.22(9)	0.53	550.40	94.29(7)	0.99	272.60
	0.400	5.50(10)	–	4.00	4.90(10)	–	4.10	7.60(10)	–	6.00	6.10(10)	–	4.40	5.70(10)	–	4.00
		218	22.12 (78.50)		224	12.59 (76.94)		218	19.92 (78.15)		227	12.27 (67.82)		218	22.12 (78.50)	