Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 273 (2025) 125–132

Procedia Computer Science

www.elsevier.com/locate/procedia

XIII Latin American Algorithms, Graphs, and Optimization Symposium (LAGOS 2025)

Generalized Capacitated Vertex Separator Problem: Models and Algorithms

Sergio Anglada[a], Carmen Galé[a], Juan José Salazar-González[b,*]

[a]Dept. de Métodos Estadísticos, Instituto Universitario de Matemáticas y Aplicaciones (IUMA), Universidad de Zaragoza, Spain
[b]Instituto Universitario de Matemáticas y Aplicaciones (IMAULL), Universidad de La Laguna, Tenerife, Spain

Abstract

Given an undirected graph and two numbers q and b, the Capacitated Vertex Separation Problem (CVSP) looks for a vertex subset of minimum cardinality such that the connected components in the subgraph generated after the vertex removal can be packed into no more than q bins of cardinality at most b. This problem has been studied in graph theory, and most of the success in solving it is due to the hypothesis that the objective function minimizes the number of deleted vertices, that is, each node removal contributes *identically* to the objective function. In our work, this hypothesis is relaxed so each vertex has a cost and a weight, and the problem aims to minimize the total cost of the removed vertices while the total vertex weight in each bin is within the given capacity b, still limiting the number of bins to at most q. We introduce several mathematical formulations for the new problem, called the Generalized Capacitated Vertex Separator Problem (GCVSP), and analyze the performance of algorithms based on such formulations.

Peer-review under responsibility of the Program Committee of LAGOS 2025.

Keywords: Graph disconnection; vertex removal; mixed integer linear programming

1. Introduction

Let $G = (V, E)$ be an undirected connected graph, where V is the set of n vertices and E is the set of edges. Given a positive integer b and a cost related to each vertex, the *Vertex Separator Problem* (VSP) consists in finding a partition of V into three nonempty subsets A, B and S such that $|A| \leq b$, $|B| \leq b$, there is no edge of E between A and B, and the total cost of the vertices in S is minimum. The subset S is called the *separator* and the subsets A and B are called *shores* (of the separator). The VSP is $\mathcal{NP}$-hard for general graphs ([5]). [2] and [11] investigated the structure of the feasible set of solutions determining some families of facet-defining inequalities. The VSP has many applications, such as the detection of brittle vertices in telecommunications networks ([6]), or in identifying the minimal separator in divide-and-conquer-based graph algorithms ([7]). [1] provide a review of exact and heuristic methods for the VSP.

[9] investigate a variant of the VSP where more than two shores are allowed. The variant assumes a pre-specified limit b on the cardinality of each shore, but not a pre-specified limit on the number of shores. This variant assume

* Corresponding author. Email: jjsalaza@ull.es

1877-0509
Peer-review under responsibility of the Program Committee of LAGOS 2025.
10.1016/j.procs.2025.10.289

identical cost for all vertices, thus the aim is to minimize the cardinal of the separator. [8] describe another cutting-plane approach to solve a related variant of the VSP where also the number of shores is limited to at most a pre-specified parameter q. The new problem is called *Capacitated Vertex Separator Problem* (CVSP), and it consists of finding the smallest separator subset such that the connected components of the resulting graph after the removal of S can be packed into no more than q shores of a cardinality at most b. The identical vertex cost assumption has a tremendous impact in the performance of a cutting-plane approach since some precedence constraints reduce equivalent solutions while keeping an optimal one, and it is a common assumption in all previous articles.

CVSP is closely related to other graph interdiction problems consisting of finding an optimal strategy for removing a limited number of vertices in a graph to maximally disconnect it, either by maximizing the number of connected components or minimizing the size of the largest connected component ([10]). CVSP is equivalent to the matrix decomposition problem studied in [4] in which two capacity constraints are imposed. Given a matrix A, a block is a submatrix formed by a subset of rows of the matrix A. Let q and b be the parameters for the maximum number of blocks and the maximum number of rows per block, respectively. The matrix decomposition problem consists of defining a set of blocks and assigning the maximum number of rows to some block such that:

- There are no more than q blocks.
- Each row is assigned to at most one block.
- Each block contains at most b rows.
- Two rows in different blocks cannot have non-zero elements in the same column.

In this work, we study a further extension of the CVSP, leading to what we call the *Generalized Capacitated Vertex Separator Problem* (GCVSP). In addition to the undirected graph $G = (V, E)$, the two numbers b and q, and a cost c_i associated with each vertex $i \in V$, we also consider a demand (or weight) p_i for each $i \in V$. If i is removed, the number c_i represents the cost in the objective function. In network segmentation and contingency planning, it is often necessary to deliberately remove a subset of vertices so that the remaining graph splits into a small number of manageable zones. Each vertex i carries a removal cost c_i (e.g., operational disruption or budget) and a demand p_i representing the load that remains served in the network. The design goal is to choose a minimum-cost separator such that the post-removal components can be assigned to at most q resources (e.g., operators or backup systems), each with capacity b in terms of total vertex weight. Equivalently, we seek a set of vertices to remove that simultaneously minimizes total cost and guarantees that, after removal, no shore exceeds total demand b and the number of shores is at most q. Allowing heterogeneous parameters c_i and p_i naturally captures differentiated removal costs and demands at vertices. Hence, p_i represents the weight for the capacity constraint in the shore where i goes in the partition. The CVSP studied in [8] is the particular case of GCVSP where $c_i = p_i = 1$, which allows the use of precedence constraints. In this paper, by allowing non-identical costs and weights, we face the difficulty of a harder combinatorial optimization problem. This work contributes with new mathematical formulations for the GCVSP. Section 2 describes a compact representative-based model, and Section 3 develops a shore-based formulation that can be tackled using a branch-and-price framework. A computational evaluation of the corresponding algorithms solving benchmark instances is presented in Section 4.

2. Representative-based formulation for the GCVSP

In most graph disconnection problems, variables are defined to implicitly or explicitly encode information about which vertices are removed or kept active, as well as the group (in the case of CVSP, the shore) to which the active vertices belong.

[4] define a binary variable based on two indices. One index represents a vertex and the other index represents a shore. The variable with indices i and k takes the value one when the vertex i is assigned to the shore k, for all $i \in V$ and $k \in \{1, \dots, q\}$. This family of variables is affected by symmetries, as any permutation of the indices $\{1, \dots, q\}$ results in an equivalent feasible solution. To address this issue, they introduced several valid inequalities, including the so-called *block-invariant inequalities*, which remain invariant under permutations of the indices $\{1, \dots, q\}$. [8]

approached the CVSP using the following binary decision variables:

$$x_i = \begin{cases} 1 & \text{if vertex } i \text{ belongs to the separator,} \\ 0 & \text{otherwise,} \end{cases} \qquad i \in V.$$

These variables have the disadvantage of needing sophisticated families of inequalities (to be managed through complex separation procedures in a branch-and-cut framework) to have a CVSP formulation.

We propose a new and compact formulation using a slightly larger set of binary variables with respect to the binary variables used in [4] and [8]. As reported in Section 4, this new formulation shows better performance than the previous formulations on some CVSP instances. In addition, the new and compact formulation can easily be presented for the GCVSP, rather than specifically for the CVSP. Given the separator subset S, we define a representative vertex for each shore. To eliminate symmetry in the formulation, the representative of each shore is chosen as the vertex with the lowest index. The binary variables are defined as follows:

$$w_i^k = \begin{cases} 1 & \text{if vertex } i \text{ belongs to the shore represented by vertex } k, \\ 0 & \text{otherwise,} \end{cases} \qquad i, k \in V \ : \ k \leq i.$$

A vertex i is the representative of its shore if and only if $w_i^i = 1$. For a vertex $i \in V$, if $w_i^k = 0$ for all $k \in V$ with $k \leq i$ then the vertex i does not belong to any shore and must therefore be part of the separator. Furthermore, compared to the formulation in [8], we have lifted the solution space by establishing the relationship $x_i = 1 - \sum_{k \leq i} w_i^k$, with the advantage that it is now a compact formulation.

Without loss of generality, we assume that all edges $\{i, j\} \in E$ satisfy $i < j$. Under this assumption, the GCVSP can be formulated as follows:

$$\max_{w} \quad \sum_{i \in V} c_i \sum_{k \leq i} w_i^k \tag{1a}$$

$$\text{subject to :} \quad \sum_{k \leq i} w_i^k \leq 1 \qquad i \in V \tag{1b}$$

$$\sum_{k \leq i} p_i w_i^k \leq b \, w_k^k \qquad k \in V \tag{1c}$$

$$\sum_{k \in V} w_k^k \leq q \tag{1d}$$

$$w_i^k - w_j^k \leq 1 - \sum_{l \leq j} w_j^l \qquad \{i, j\} \in E, \ k \in V : k \leq i \tag{1e}$$

$$w_j^k - w_i^k \leq 1 - \sum_{l \leq i} w_i^l \qquad \{i, j\} \in E, \ k \in V : k \leq i \tag{1f}$$

$$w_i^k \in \{0, 1\} \qquad i \in V, \ k \in V : k \leq i. \tag{1g}$$

The objective function (1a) maximizes the cost of the non-removed vertices assigned to a representative vertex, that is, the vertices that remain active. Constraints (1b) impose that each vertex is either the representative of its shore ($w_i^i = 1$) or, it is assigned to at most one representative, or, if none of the variables takes the value 1, it is removed. Constraints (1c) ensure the capacity restriction of the shores. In the case of a vertex $k \in V$ such that $w_k^k = 0$, that is,

it is not the representative of the shore, this constraint guarantees that no vertex $i \in V$ can be assigned to vertex k. Constraint (1d) imposes that the number of shores, that is, the number of representatives, cannot exceed q. Constraints (1e) and (1f) ensure that two vertices forming an edge share the same representative if both are active. Constraints (1g) impose the binary character of the variables w_i^k. Observe that the model can also be adapted to deal with a further generalization of the GCVSP where each bin $l \in \{1, \dots, q\}$ could have a different capacity b^l, or even the limit q is not on the number of bins but a weighted function.

Finally, as observed in [4], there is a big advantage in the assumption that all vertex removal has the same cost in the objective function. Indeed, in this case, there are many solutions with the same cost. This fact allows reducing the feasible region with some precedence conditions between removing the vertices, as in [4]. As explored also in [8], if the neighborhood $N(j)$ of a vertex j is strictly contained within the neighborhood $N(i)$ of a vertex i, one can enforce that vertex j is removed only if vertex i was removed. This claim is based on the fact that any feasible solution where $x_j = 1$ and $x_i = 0$ can be transformed into an equivalent feasible solution of the same cost where $x_j = 0$ and $x_i = 1$. These precedence constraints can be formalized as $x_j \leq x_i$ for all $i, j \in V$ with $N(j) \setminus \{i\} \subset N(i) \setminus \{j\}$. Moreover, when two vertices have identical neighbourhoods, one can establish a hierarchy in the removal of these vertices. The aim is to eliminate symmetries by ensuring that the vertex with the smaller index can only be removed once the vertex with the larger index has already been removed. It can be forced in the mathematical formulation as $x_j \leq x_i$ for all $i, j \in V$ with $j < i$ and $N(j) \setminus \{i\} = N(i) \setminus \{j\}$. This way, there will always be an optimal solution that verifies these constraints, and equivalent feasible solutions may be excluded. Note that this reduction of the solution space can also be applied to CVSP using Model (1) because $x_i = 1 - \sum_{k \leq i} w_i^k$ and $c_i = p_i = 1$ for all $i \in V$. The same idea can be applied to GCVSP instances by additionally requiring that, when enforcing precedence constraints, the cost of vertex i does not exceed the cost of vertex j, i.e., $c_i \leq c_j$. This ensures that any transformation of a feasible solution respecting the precedence constraints does not increase the total cost. For general cost values, this precedence constraint does not have the tremendous impact that it has on CVSP instances.

3. Shore-based formulation

This section introduces an alternative approach, based on a large number of columns, i.e. suitable to be used within a branch-and-price framework. Let $\mathcal{R} = \{R \subseteq V : \sum_{i \in R} p_i \leq b\}$ be the set of all possible shores in the solution, and let $\mathcal{R}_i$ denote all the subsets in $\mathcal{R}$ including vertex i for each $i \in V$. We define the following binary variables:

$$z^R = \begin{cases} 1 & \text{if the set } R \text{ corresponds to a shore in the solution,} \\ 0 & \text{otherwise,} \end{cases} \qquad R \in \mathcal{R}.$$

Notice that variables z^R are exponential in number. A shore-based formulation for GCVSP is:

$$\max_{z} \quad \sum_{i \in V} c_i \sum_{R \in \mathcal{R}_i} z^R \tag{2a}$$

$$\text{subject to} \quad \sum_{R \in \mathcal{R}_i} z^R \leq 1 \qquad i \in V \tag{2b}$$

$$\sum_{R \in \mathcal{R}_i \cup \mathcal{R}_j} z^R \leq 1 \qquad \{i, j\} \in E \tag{2c}$$

$$\sum_{R \in \mathcal{R}} z^R \leq q \tag{2d}$$

$$z^R \in \{0, 1\} \qquad R \in \mathcal{R}. \tag{2e}$$

Constraints (2b) ensure that each vertex either is removed or belongs to a shore in the solution. Constraints (2c) impose that two vertices forming an edge in the graph cannot belong to different shores. Constraints (2d) limit the number of shores. Finally, (2e) are the integrability constraints.

The large number of variables in this formulation requires solving the so-called pricing problem. Indeed, the above formulation can be solved following a branch-and-price scheme, which means a branch-and-bound framework where at each node one must solve a sequence of incomplete master problems. Each incomplete master problem is defined by only some variables. Once the linear-programming relaxation of an incomplete master problem was solved, and an optimal dual solution is available, the pricing problem aims at finding a missing variable that should be incorporated into the master problem and potentially improve the linear-programming bound. Let η_i (≥ 0) related to each equation in (2b) for $i \in V$, and λ_e (≥ 0) related to each inequality in (2c) for $e \in E$, be the dual optimal solution of the current master problem. The pricing problem can be formulated with the following binary variables:

$$x'_i = \begin{cases} 1 & \text{if } i \text{ belongs to } R, \\ 0 & \text{otherwise,} \end{cases} \qquad i \in V,$$

$$y'_e = \begin{cases} 1 & \text{if at least a vertex in } e \text{ belongs to } R, \\ 0 & \text{otherwise,} \end{cases} \qquad e \in E,$$

by the following 0-1 linear program:

$$\begin{aligned} \max_{x',y'} \quad & \sum_{i \in V} \eta_i x'_i + \sum_{e \in E} \lambda_e y'_e \\ \text{subject to :} \quad & \sum_{i \in V} p_i x'_i \leq b \\ & 0 \leq x'_i \leq y'_e \leq 1 \qquad e \in E,\ i \in e \\ & x'_i \in \{0,1\} \qquad i \in V. \end{aligned}$$

[3] observed that, when $p_i = 1$ for all $i \in V$, the pricing problem is $\mathcal{NP}$-hard as the max-clique problem is a particular case of it. Without the b limitation, the relaxed pricing problem is a min-cut problem and therefore can be solved in polynomial time.

4. Computational experience

We aim to show the performance of a modern MILP solver on GCVSP models (1) and (2). The computational experiments were performed on a personal computer with 13th Gen Intel Core i9-13900F at 2.0 GHz having 64.0 GB of RAM, and Windows 11 64-bit as the operating system, using ILOG CPLEX Studio 22.1.0 and programming in a C++ API of ILOG Concert Technology. The time limit is set to 1800 seconds.

We illustrate the performance of our proposal on two computational experiments. In a first experiment, we use Model (1) to solve CVSP instances for which we also run the best implementation described in [8], the current state-of-the-art CVSP approach. We thanks the authors for having provided us with their computer implementation. In a second experiment, we evaluate Model (1) on proper GCVSP instances (i.e. with non-identical vertex costs and weights).

For the first experiment, we consider ten DIMACS instances originating from graph coloring problems. The number of vertices in these graphs ranges from 23 to 138. These CVSP instances were considered in [4] and [8] assuming a given number of shores $q \in \mathbb{N}$ and the capacity limit defined by $b = \lceil |V|/q \rceil$. To better explore the effect of b and q, for a given value of q, we have extended the benchmark collection of CVSP instances with $b = \lceil \beta \cdot |V|/q \rceil$ using $\beta \in \{0.5, 0.7, 0.9, 1.0, 1.1\}$. For cases with repeated values of b for the same value of q, they have been manually approximated to have five different values of b for each value of q. Note that the fourth value of our configuration of the values of b matches the configuration set in [8]. Finally, the values of q considered are $\{2, 4, 6, 8\}$. Table 1 shows the values of b and q considered for each of the ten DIMACS graphs, making a total of 200 different instances.

Table 1. Instance information for the CVSP experiment.

Graph	$\|V\|$	$\|E\|$	b			
			$q = 2$	$q = 4$	$q = 6$	$q = 8$
myciel4	23	71	{6, 9, 11, 12, 13}	{3, 4, 5, 6, 7}	{2, 3, 4, 5, 6}	{1, 2, 3, 4, 5}
queen5_5	25	160	{7, 9, 12, 13, 14}	{3, 4, 5, 6, 7}	{2, 3, 4, 5, 6}	{1, 2, 3, 4, 5}
queen6_6	36	290	{9, 13, 17, 18, 20}	{5, 7, 8, 9, 10}	{3, 5, 6, 7, 8}	{2, 3, 4, 5, 6}
myciel5	47	236	{12, 17, 22, 24, 26}	{6, 9, 11, 12, 13}	{4, 6, 7, 8, 9}	{3, 4, 5, 6, 7}
queen7_7	49	476	{13, 18, 23, 25, 27}	{7, 9, 12, 13, 14}	{5, 6, 7, 8, 9}	{4, 5, 6, 7, 8}
huck	74	301	{19, 26, 34, 37, 41}	{10, 13, 17, 19, 21}	{7, 9, 12, 13, 14}	{5, 7, 9, 10, 11}
jean	80	254	{20, 28, 36, 40, 44}	{10, 14, 18, 20, 22}	{7, 10, 12, 14, 15}	{5, 7, 9, 10, 11}
david	87	406	{22, 31, 40, 44, 48}	{11, 16, 20, 22, 24}	{8, 11, 14, 15, 16}	{6, 8, 10, 11, 12}
myciel6	95	755	{24, 34, 43, 48, 53}	{12, 17, 22, 24, 27}	{8, 12, 15, 16, 18}	{6, 9, 11, 12, 14}
anna	138	493	{35, 49, 63, 69, 76}	{18, 25, 32, 35, 38}	{12, 17, 21, 23, 26}	{9, 13, 16, 18, 19}

Table 2. Performance when solving the CVSP instances in Table 1.

β	*#Inst*	*Approach*	*#Solved*	*Avg Time*	P_{25}	P_{50}	P_{75}
0.5	40	Furini et al.	13 (32.5%)	87.58	0.16	0.63	9.19
		Model (1)	38 (95.0%)	75.48	0.28	1.15	9.61
0.7	40	Furini et al.	14 (35.0%)	42.86	0.17	1.61	5.63
		Model (1)	36 (90.0%)	124.70	0.68	2.46	17.94
0.9	40	Furini et al.	28 (70.0%)	33.66	0.05	0.15	2.01
		Model (1)	37 (92.5%)	212.26	1.76	15.48	115.60
1.0	40	Furini et al.	28 (70.0%)	22.57	0.05	0.15	1.45
		Model (1)	39 (97.5%)	197.47	2.33	14.44	132.20
1.1	40	Furini et al.	28 (70.0%)	127.26	0.05	0.14	2.76
		Model (1)	39 (97.5%)	216.34	1.94	9.95	163.45
all	200	Furini et al.	111 (55.5%)	61.95	0.05	0.19	3.44
		Model (1)	189 (94.5%)	165.75	0.78	6.16	71.08

We have applied two approaches to each CVSP instance: Configuration C in [8], and the representative-based model (1) described in Section 2. Since $c_i = p_i = 1$ for each vertex i, we have activated the precedence constraints to help both approaches. Configuration C is the best basic variant of the branch-and-cut algorithm in [8] in the sense that it solves to optimality the most instances within a limited time. The performance is summarized in Table 2. Column *#Inst* shows the number of instances in each case, while column *#Solved* reports how many of them were solved to optimality by each approach within the time limit. *Avg Time* indicates the average time (in seconds) for the solved instances, and P_{25}, P_{50} and P_{75} show the times corresponding to the fastest 25%, 50% and 75% of solved instances, respectively.

Across all cases, Model (1) consistently maintains a high success rate, solving 94.5% of instances overall. In contrast, the approach in [8] shows a notable difference between the two smallest values of b and the three larger ones. For $\beta = 0.5$ and $\beta = 0.7$, [8] solves only 32.5% and 35% of instances, respectively. However, when b takes values closer to $\lceil |V|/q \rceil$, the performance of [8] improves significantly, solving 70% of instances for the three largest values

Table 3. Instance information for the GCVSP experiment.

Graph	$\|V\|$	$\|E\|$	P	b			
				$q=2$	$q=4$	$q=6$	$q=8$
myciel4	23	71	142	71	36	24	18
queen5_5	25	160	320	160	80	54	40
queen6_6	36	290	580	290	145	97	73
myciel5	47	236	472	236	118	79	59
queen7_7	49	476	952	476	238	159	119
huck	74	301	602	301	151	101	76
jean	80	254	508	254	127	85	64
david	87	406	812	406	203	136	102
myciel6	95	755	1510	755	378	252	189
anna	138	493	986	493	247	165	124

of b. It is worth observing that the performance of [8] degrades when β reduces to 0.5, but these instances were not considered in [8]. Moreover, the optimal solutions found by the approach in [8] are contained within the set of optimal solutions obtained by Model (1) for all values of β, except for the value 0.7, where there exist two instances that are solver to optimality by [8] but not by Model (1).

In addition, for the CVSP instances solved by the approach in [8], the reported computational times are often significantly shorter than those required by Model (1). However, these times are not directly comparable, as they correspond to different sets of instances (only those solved to optimality by each approach). Notably, since Model (1) successfully solves a larger number of instances, it can be understood that it tackles relatively more difficult instances. Consequently, the observed patterns reflect a trade-off: [8] demonstrates higher computational efficiency on the instances it solves, whereas Model (1) provides broader coverage, achieving optimality on a greater portion of instances within the time limit. Furthermore, for instances not solved to optimality by either approach, the representative-based model consistently yields higher-quality solutions: the average GAP is 20.70 compared to 38.30 for [8], and the maximum GAP is 37.50 versus 82.30, highlighting that even when optimality is not reached, Model (1) produces significantly better solutions.

For the second experiment, we constructed GCVSP instances based on the DIMACS graphs previously used for the CVSP. To define the cost c_i and weight p_i of each vertex i, we set both equal to the degree of vertex i. Since introducing weights changes the effective capacity of each shore, we adjusted the value of b accordingly. In analogy with the CVSP configuration, we set b equal to the total vertex weight in the graph divided by the number of shores, i.e. $b = \lceil P/q \rceil$ where $P = \sum_{i \in V} p_i$. This ensures that each shore can accommodate a balanced total vertex weight without requiring the removal of an excessive number of vertices. The values of q were kept consistent with the CVSP experiment, $q \in \{2, 4, 6, 8\}$, producing 40 different GCVSP instances in total. Note that, since each vertex weight is set equal to its degree, the total vertex weight P is always equal to twice the number of edges in the graph, $P = 2|E|$. Table 3 summarizes the instances used in the second experiment.

Table 4 shows the results of solving each GCVSP instance in Table 3 with Model (1). For each instance, the table shows the objective value *Obj* of the best GCVSP solution found, the gap *GAP* reported by the MILP solver, and the computational time *Time* in CPU seconds, replaced by "-" when the MILP solver reached the time limit (1800 seconds). The best lower bound found by the MILP solver on an instance is $Obj \cdot (1 - GAP/100)$. The MILP solver found the optimal solution to 70% of the instances, with an average computational time of 199 seconds on the 28 solved instances. We have also performed experiments on using Model (2), but our branch-and-price implementation showed very low performance and was unable to solve even instances with 25 vertices. Clearly, further investigations in pricing techniques and additional details are needed to improve our branch-and-price code, and we leave this task for future work.

Table 4. Performance when solving the GCVSP instances in Table 3 with Model (1).

Graph	$q = 2$			$q = 4$			$q = 6$			$q = 8$		
	Obj	*GAP*	*Time*	*Obj*	*GAP*	*Time*	*Obj*	*GAP*	*Time*	*Obj*	*GAP*	*Time*
myciel4	83	0	0.42	67	0	0.80	66	0	0.57	61	0	0.22
queen5_5	168	0	1.40	104	0	3.05	100	0	2.35	78	0	2.15
queen6_6	305	0	12.39	196	0	158.50	175	0	138.00	145	0	107.60
myciel5	279	0	12.91	214	0	168.30	210	0	33.06	199	0	29.46
queen7_7	496	0	164.90	316	0	1048.00	270	60.81	-	234	64.51	-
huck	508	0	8.71	454	0	42.08	402	0	110.60	380	0	150.00
jean	409	0	36.99	369	0	96.38	341	0	104.70	308	0	1491.00
david	561	0	266.80	509	0	1379.00	462	11.48	-	429	18.10	-
myciel6	858	34.68	-	660	58.14	-	620	48.07	-	586	53.83	-
anna	689	19.73	-	582	48.93	-	550	55.31	-	542	55.07	-

Acknowledgment

This research has been funded by the Spanish Government under grants PID2022-139543OB-C43, PID2023-148599NB-I00 and PCI2024-155092-2, and by the *Gobierno de Aragón* under grant E41-23R.

References

[1] Althoby, H.Y., Didi Biha, M., Sesboüé, A., 2020. Exact and heuristic methods for the vertex separator problem. Computers & Industrial Engineering 139, 106135.
[2] Balas, E., de Souza, C., 2005. The vertex separator problem: a polyhedral investigation. Mathematical Programming , 583–608.
[3] Bastubbe, M., Lübbecke, M., 2020. A branch-and-price algorithm for capacitated hypergraph vertex separation. Mathematical Programming Computation 12, 39–68.
[4] Borndörfer, R., Ferreira, C.E., Martin, A., 1998. Decomposing matrices into blocks. Society for Industrial and Applied Mathematics 9, 236–269.
[5] Bui, T.N., Jones, C., 1992. Finding good approximate vertex and edge partitions is $\mathcal{NP}$-hard. Information Processing Letters 42, 153–159.
[6] Didi Biha, M., Meurs, M.J., 2011. An exact algorithm for solving the vertex separator problem. Journal of Global Optimization 49, 425–434.
[7] Evrendilek, C., 2008. Vertex separators for partitioning a graph. Sensors 8, 635–657.
[8] Furini, F., Ljubić, I., Malaguti, E., Paronuzzi, P., 2021. Casting light on the hidden bilevel combinatorial structure of the capacitated vertex separator problem. Operations Research 70, 2399–2420.
[9] Oosten, M., Rutten, J., Spieksma, F., 2007. Disconnecting graphs by removing vertices: A polyhedral approach. Statistica Neerlandica 61, 35–60.
[10] Shen, S., Smith, J., 2012. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. Networks 60, 103–119.
[11] de Souza, C., Balas, E., 2005. The vertex separator problem: algorithms and computations. Mathematical Programming 103, 609–631.