XI Balkan Conference on Operational Research
BALCOR 2013
Belgrade, Zlatibor, 07-10 September 2013
UNIVERSITY OF BELGRADE
FACULTY OF ORGANISATIONAL SCIENCES

Computing Graph Separators with Variable Neighborhood Search

David Fernández-López[1], Jesús Sánchez-Oro[1], Abraham Duarte[1]

[1] Dept. Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain, {david.fernandez, jesus.sanchezoro, abraham.duarte}@urjc.es

Abstract: *Given an undirected and connected graph $G = (V,E)$ and a bounded positive integer b, the Vertex Separator (VS) problem consists of finding the set of vertices C, whose removal divides G into two separated components A,B, where $\max\{|A|,|B|\} \leq b$. This NP-hard optimization problem appears in a wide range of applications. For instance, in telecommunication networks, a separator determines the capacity and brittleness of the network. In the field of graph algorithms, the computation of balanced small-sized separators is very useful, especially for divide-and-conquer algorithms. In bioinformatics and computational biology, separators are required in grid graphs providing a simplified representation of proteins. This paper presents a new heuristic algorithm based on the Variable Neighborhood Search (VNS) methodology for computing vertex separators. We compare our procedure with the state-of the-art methods (two branch-and-bound procedures recently published). Computational results show that our procedure obtains the optimum solution in all of the small and medium instances, and obtains high-quality results in large instances. Although the branch-and-bound methods assess the optimality of the solution found, they need large computing time. On the other hand, our procedure obtains almost all optima in a small fraction of that time (without assessing the optimality of the solution found).*

Keywords: *Combinatorial optimization, Metaheuristics, VNS, Graph separators*

1. INTRODUCTION

Consider a graph $G = (V,E)$, where V is the set of vertices and E the set of edges. Let c_j be the cost associated with each vertex $j \in V$. A separator in G is usually defined as a subset of vertices or edges whose removal disconnect the graph. Separators are also known by other names, including bisectors, bifurcators, balanced cuts, and partitions.

In this paper we particularly focus on the Vertex Separator (VS) problem, which consists of finding a partition of V into three non-empty subsets A,B and C, such that there is not any edge between A and B, the size of both sets are bounded by a positive integer b (which is usually a function of $|V|$), and the sum of the weight of the vertices in C is minimized.

A solution x of the VS problem for a given graph G can be represented as three sets A,B, and C such that $A \cup B \cup C = V$ and $A \cap B = A \cap C = B \cap C = \emptyset$. Therefore, the value of the objective function, Sep, of a solution, $x = \{A,B,C\}$, is defined as $Sep(x,G) = \sum_{j \in C} c_j$.

The optimization problem can be formulated as follows:

$$\begin{aligned} \min \; & \sum_{j \in C} c_j \\ s.t. \; & \\ & \max\{|A|,|B|\} \leq b \\ & \forall i \in A \wedge \forall j \in B, (i,j) \notin E \end{aligned}$$

Notice that the VS problem is completely equivalent to maximizing the sum of the weight of the vertices in A and B (see for instance [**?**]).

Figure **??**.(a) shows an illustrative example of a graph G with five vertices and six edges. Figure **??**.(b) depicts a possible solution, where sets A,B, and C are represented as dashed rectangles. If we assume, for the sake of simplicity, that each vertex has a cost of 1, the value of this solution is 1 since there is only one vertex in the set C.

This optimization problem was originally introduced in [**?**] in the context of VLSI design. However, finding balanced separators of small size become an important task in several contexts. For instance, in telecommunication networks, a separator determines the capacity and brittleness of the network ([**?**, **?**]). In the field of graph algorithms, the computation of balanced small-sized separators is very useful, especially for divide-and-conquer algorithms (see [**?**] for a larger description). In bioinformatics and computational biology, separators are required in grid graphs providing a simplified representation of proteins.

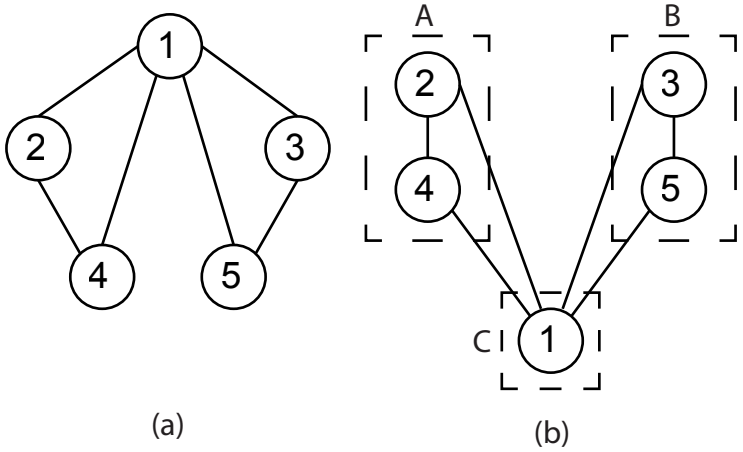

Figure 1 Example of a graph (a) and a possible Vertex Separator solution (b)

Finding the minimum vertex separator of a general graph is a NP-hard problem [**?**]. Therefore, exact approaches only solve instances of moderate size. In particular, Souza and Balas [**?**] proposed a mixed integer programming formulation, and investigated the VS polytope and the convex hull of incidence vectors of vertex separators. This theoretical study was afterwards included in a branch-and-cut procedure. Computational results showed that the exact method was able to optimally solve instances with size ranging from 50 to 150 in moderate computing time. Biha and Meurs [**?**] introduced new classes of valid inequalities and a simple lower bound. Computational experiments showed that the new exact procedure was able to solve to optimallity all the instances introduced in Balas and Souza [**?**] in shorter computing time. Therefore, this method emerges as the state-of-the-art regarding exact procedures.

Much of the previous work on the VS problem is based on designing approximation algorithms. Lipton and Tarjan [**?**] showed that an n-vertex planar graph has a balanced separator of size $O(\sqrt{n})$. However, most of the papers approximate the VS problem by solving the Edge Separator (ES) problem. In particular, these approaches first design an approximation algorithm for the ES problem and then adapt this algorithm to the VS problem. See for instance, Leighton and Rao [**?**] or Arora et al. [**?**] with an approximation ratio of $O(\log n)$ and $O(\sqrt{\log n})$, respectively. This approach works well on graphs with bounded degree. However, for general graphs, the situation is completely different (Feige et al. [**?**]). The best approximation algorithm for the VS problem was introduced in Feige et al. [**?**] . In particular, they proposed linear and semidefinite program relaxations, and rounding algorithms for these programs. They obtained an $O(\sqrt{\log(opt)})$ approximation, where opt is the size of the optimal separator.

Despite the fact that this problem has been the subject of extensive research, it has been mainly ignored in the heuristic and metaheuristic community. As far as we now, there is only one heuristic method introduced in Souza and Balas [**?**]. In particular, the unique objective of this method is to generate feasible solutions for the branch-and-cut procedure. The method first solves the linear relaxation of the original problem. Then, starting from the fractional solution, it constructs an integer solution by sequentially rounding up (when it is possible) the fractional variables using a greedy function. After each rounding up, some variables are rounded down to 0 to fulfill the constraints of the problem.

In this paper, we propose a Variable Neighborhood Search (VNS) approach [**?**] for the Vertex Separator problem. This metaheuristic has been successfully applied to a large number of optimization problems. See for instance [**?**, **?**, **?**, **?**, **?**]. We particularly exploit the advantages of the Reduced Variable Neighborhood Search (RVNS) variant to design an efficient and effective solving procedure for the VS problem. Section 2 introduces the constructive procedure designed for this problem. Section 3 describes a new shake procedure, designated to balance intensification and diversification. Section 4 is devoted to describe the VNS procedure itself, and how it uses the constructive and the shake procedures. Section 5 reports on an extensive computational experience to validate the proposed algorithm by comparing its performance and computing time with the state-of-the-art methods. Finally, Section 6 summarizes the main conclusions of our research.

2. CONSTRUCTIVE PROCEDURE

Level algorithms (see [**?**]) are constructive procedures based on the partition of the vertices of a graph in different levels, $L_1, \ldots, L_s$, such that the endpoints of each edge in the graph are either in the same level L_i or

```
1: procedure Constructive(T)
2: T = {L_1, L_2, ..., L_s}
3: limit ← ⌊s/2⌋
4: i ← 1
5: j ← s
6: fullA ← false
7: fullB ← false
8: while (i < limit) or (j > limit) do
9:    if (|A| + |L_i| < b) and not fullA then
10:      A ← A ∪ L_i
11:   else
12:      fullA ← true
13:      critical ← i
14:      limit ← i + 1
15:   end if
16:   if (|B| + |L_j| < b) and not fullB then
17:      B ← B ∪ L_j
18:   else
19:      fullB ← true
20:      critical ← j
21:      limit ← j − 1
22:   end if
23:   i ← i + 1
24:   j ← j − 1
25: end while
26: for all v ∈ L_critical do
27:   N(v) → {u ∈ V : (u, v) ∈ E}
28:   N_A(v) → N(v) ∩ A
29:   N_B(v) → N(v) ∩ B
30:   if |N_A(v)| = ∅ then
31:      B → B ∪ {v}
32:   else if |N_B(v) = ∅ then
33:      A → A ∪ {v}
34:   else
35:      C → C ∪ {v}
36:   end if
37: end for
```

Algorithm 1: Pseudocode of Constructive procedure

in two consecutive levels, L_i and L_{i+1}. This level structure guarantees that vertices in alternative levels are not adjacent. Our constructive procedure generates the level structure by using an Breadth-First Search (BFS) method. This algorithm receives as input parameter a vertex v. The method then constructs a spanning tree rooted at v. Therefore, if a graph has $|V|$ vertices, we can construct $|V|$ different spanning trees.

Algorithm **??** shows the pseudo-code of our constructive procedure. In particular, it receives as input parameter an spanning tree T and returns a feasible solution of the VS problem. The algorithm starts by identifying the set of levels of the tree (step 2). We then initialize a set of auxiliary variables (steps 3 to 7). The algorithm tries to alternatively assign one level of the tree to each set. In particular, level L_1 is assigned to A, then level L_s is assigned to B. If both sets are not full, we perform another iteration assigning L_2 and L_{s-1} to A and B, respectively (see steps 9 to 24). This logic is maintained until we reach the medium level ($\lfloor s/2 \rfloor$) or one of the subsets is filled (tested at steps 9 and 16). If neither A nor B reaches the maximum capacity, the critical level is $\lfloor s/2 \rfloor$. Otherwise, the critical level is the one that fills the corresponding set (see steps 13 and 20). Once a set is complete, the other set is filled with the remaining levels but the critical.

The elements in the critical level are processed in steps 26 to 37. In particular, for each vertex, we first test its adjacent vertices in A or B (steps 28 and 29). If the corresponding vertex has not adjacent in A, it can be assigned to B or viceversa (steps 30 to 33). Finally, if the vertex has adjacents in A and B, it is assigned to C.

```
1: procedure Shake(x,k)
2: x = {A,B,C}
3: Cand ← RandomlySelect(A∪B,k)
4: A ← A \ Cand
5: B ← B \ Cand
6: for all c ∈ C do
7:    feasA ← Feasible(A∪{c})
8:    feasB ← Feasible(B∪{c})
9:    if (feasA and feasB) then
10:      if ((|A| < |B|)) then
11:         A ← A∪{c}
12:      else
13:         B ← B∪{c}
14:      end if
15:   else if feasA then
16:      A ← A∪{c}
17:   else if feasB then
18:      B ← B∪{c}
19:   end if
20: end for
21: for all c ∈ Cand do
22:   feasA ← Feasible(A∪{c})
23:   feasB ← Feasible(B∪{c})
24:   if (feasA and feasB) then
25:      if ((|A| < |B|)) then
26:         A ← A∪{c}
27:      else
28:         B ← B∪{c}
29:      end if
30:   else if feasA then
31:      A ← A∪{c}
32:   else if feasB then
33:      B ← B∪{c}
34:   else
35:      C ← C∪{c}
36:   end if
37: end for
```

Algorithm 2: Pseudocode of Shake method

3. SHAKE METHOD

There are several optimization problems where the structure of the solution barely allows the designing of an efficient local search. The VS problem falls in this category. In particular, a solution x is defined as three disjoint subsets A, B, and C. In that solution moves become quite complex. For instance, removing one vertex in A (symmetrically in B) and inserting it in B (symmetrically in A) does not change the objective function. In this line, removing vertices from A or B and inserting them in C deteriorates the objective function. Finally, removing a vertex from C and inserting it in A or B is hardly ever possible since the new solution is usually unfeasible.

As a consequence, designing a VNS method for the VS problem then requires to implement "intelligent" shake procedures. Notice that most of the VNS algorithms use the shake function as a strategy to escape from local optima by randomly perturbing the incumbent solution. This strategy has been successfully used in several contexts ([**?**, **?**]). However, as it was aforementioned this option is not adequate for the VS problem. Therefore, the shake procedure not only must diversify but also intensify the search.

The shake function proposed in this paper, $Shake(x,k)$ is shown in Algorithm **??**. It first identifies the three disjoint sets A, B, and C (step 2). Then, we identify the set of k vertices (called $Cand$) which will be perturbed. In order to diversify the search, those vertices are selected from $A \cup B$ at random (step 3). Then, the method generates a partial solution by removing the selected k vertices (steps 4 and 5). On that partial solution, we

```
 1: procedure RVNS(x, k_max, t_max)
 2: repeat
 3:    k ← 1
 4:    repeat
 5:       x' ← Shake(x, k)
 6:       if Sep(x', G) < Sep(x, G) then
 7:          x ← x'
 8:          k ← 1
 9:       else
10:          k ← k + 1
11:       end if
12:    until k = k_max
13:    t ← CPUTime()
14: until t > t_max
15: end RVNS
```

Algorithm 3: Pseudocode of RVNS

try to reduce the objective function by removing elements from C and inserting them in A or B (intensification stage). We first test if removing one element of C and inserting it in A (step 7) or B (step 8) is feasible. We consider the function `Feasible`$(A \cup \{c\})$, which returns `true` if the insertion of c does not exceed the bound b (i.e., $A \cup \{c\} \leq b$) and $N_B(c) = \emptyset$. The functionality of `Feasible`$(B \cup \{c\})$ is completely symmetric.

The insertion of one element in either A or B can be separated in three different cases (steps 9 to 19). If it is possible to insert the element in both sets, we insert it in the set with the minimum cardinality to have roughly equally-sized sets (steps 10 to 14). Otherwise, we insert the element in the only set which generates a feasible solution (steps 15 to 19).

Finally, the shake procedure ends by inserting the k removed elements, first, in A or B (again, maintaining the size of the sets as equal as possible). When it is not possible, the element is finally inserted in C (see steps 21 to 37).

4. REDUCED VARIABLE NEIGHBORHOOD SEARCH

Variable Neighborhood Search (VNS) is a methodology for solving optimization problems based on changing neighborhood structures. In recent years, a large amount of VNS variants have been proposed. Just to mention a few of them: Variable Neighborhood Descent (VND), Reduced VNS (RVNS), Basic VNS (BVNS), Skewed VNS (SVNS), General VNS (GVNS) or Reactive VNS. We refer the reader to [**?**, **?**] for excellent reviews of this methodology.

Let N_k with $1 \leq k \leq k_{max}$ be a finite set of pre-selected neighborhood structures, where $N_k(x)$ is the set of neighbor solutions of x in the $k-$th neighborhood and k_{max} is the largest neighbor that is going to be explored. When solving an optimization problem by using different neighborhoods structures, the VNS methodology proposes to explore them in three different ways: (i) random, (ii) deterministic, or (iii) mixed, which hybridizes both, deterministic and random. In this paper, we focus on the RVNS variant, which consists of exploring (generating) solutions at random in each N_k neighborhood. This variant does not consider the application of a local search procedure. In fact, the values of the generated solutions are directly compared with the value of the incumbent solution, updating the best solution in case of improvement.

The pseudo-code of RVNS is shown in Algorithm **??**. It has three input arguments: the initial solution (generated with the constructive method proposed in Section 2), the largest predefined neighborhood (k_{max}), and the maximum computing time (t_{max}). The RVNS begins the search in the first neighborhood structure (step 3). Then, the incumbent solution is perturbed using the function `Shake`, in step 5, obtaining a new solution x' (see Section 3). In step 6, it is decided whether the RVNS performs a move, which implies that x' is better than x, or not. If so, the incumbent solution is then updated (step 7) and the method resorts to the first neighborhood (step 8). Otherwise, (i.e., x' is worse than x) RVNS explores a larger neighborhood by increasing k (step 10). Steps 5 to 11 are repeated until k_{max} is reached. This parameter determines the maximum number of different neighborhoods to be explored in the current iteration when there is no improvement in the solution. Finally, steps 3 to 13 are repeated until t_{max} is reached.

In order to diversify the search the RVNS is executed $|V|$ independent iterations. Specifically, it considers in each iteration a different spanning tree constructed with a Depth-First Search algorithm. The whole method then returns the best solution found.

5. COMPUTATIONAL EXPERIMENTS

This section reports the computational experiments that we have performed for testing the efficiency of the proposed RVNS for solving the VS problem. The algorithm was implemented in Java SE 6. We have considered four set of instances previously used in this problem. DIMACS instances come from the DIMACS challenge on graph coloring. The graphs included in this set do not have more than 150 vertices, with the exception of the instance myciel7 which has 191 vertices.

The second class of instances is derived from the Matrix Market graphs (MM graphs for short). This collection consists of a set of standard test matrices $M = M_{uv}$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The graphs are derived from these matrices by considering an edge (u, v) for every element $M_{uv} \neq 0$.

Three categories of MM graphs were generated. The first one, called MM-I, corresponds to all matrices with 20 to 100 columns. The second category, called MM-II, was obtained from the matrices whose number of columns range from 100 to 200. The third category of instances, denoted by MM-HD, only contains graphs with at least 35% of density.

We compare the results of our RVNS with those obtained by De Souza and Balas [**?**] and Biha and Meurs [**?**]. In our opinion, the performance of the computers where the algorithms were executed does not differ very much. We provide the main characteristics of the computers to facilitate the computing time comparison.

- De Souza and Balas (SB) [**?**]: Pentium 4 processor, with 2.5 GHz and 2 GB of memory.
- Biha and Meurs (BM) [**?**]: Pentium M740 processor, with 1.73 GHz and 1 GB of memory.
- Fernandez-Lopez et al. (RVNS): Intel Core 2 Duo T6400 processor, with 2.0 GHz and 3 GB of memory.

Tables **??**, **??**, **??**, and **??** report the experimental results, where each line corresponds to one specific instance. In all these tables, the first column contains the instance name (Instance) followed, in parenthesis, by the number of vertices of the graph and the density. The second column contains the optimal solution value (Opt.), the third and fourth columns give, respectively, the running times in seconds of both exact procedures (SB Time and BM Time). Finally, the fifth and the sixth columns contain the value of the objective function found by our procedure (RVNS) and its associated computing time in seconds (Time).

The easiest instances are those in set MM-I, reported in Table 1. Our RVNS procedure matches all optima in 0.13 seconds while BM and SB need 7.87 seconds and 48.46 seconds, respectively. Obviously the other two methods guarantees the optimality of the solution, but employ much more CPU time.

Instances reported in Table 2 have a moderate difficulty to be solved. As we can see our RVNS maintains a really short computing time (1.43 seconds on average), which compares favorably with those obtained by SB (147.92 seconds) and BM (53.98 seconds). Our method finds the optimal solution in 19 instances (out of 20). In the only instance where the RVNS does not match the optimal solution, it finds a solution one unit below the optimum.

Table 3 contains the most difficult instances in the set MM-HD. In this case, our method does not match the optimum solution in 6 instances (out of 39). However, the average objective function attained by the RVNS (72.21) is extremely close to the average of the optima (72.49). Regarding the CPU time, our method again exhibits a remarkable performance (0.46 seconds on average) compared with SB (13.74 seconds) and BM (13.67 seconds).

Table 4 reports the results of the tree methods over a different set of instances. In particular, those instances obtained from the DIMACS challenge on graph coloring. Again, our method presents outstanding results, obtaining the optimal solution in all instances but one. In addition, in that instance, it obtain a solution one unit below the optimum. The RVNS needs only about one second to find these results, while its competitors consumes about 600 seconds (obviously assessing that the solution found is the optimum).

Table 1: Results on the MM-I instances.

Instance	Opt.	SB Time	BM Time	RVNS	Time
ash219 (85, 0.06)	79	0.18	1.19	79	0.32
ash85 (85, 0.17)	72	2.87	8.61	72	0.35
bcspwr01 (39, 0.16)	36	0.02	0.13	36	0.02
bcspwr02 (49, 0.15)	44	0.06	0.48	44	0.05
bcsstk01 (48, 0.55)	30	1.63	0.06	30	0.03
bfw62a (62, 0.34)	59	0.05	0.58	59	0.11
can24 (24, 0.57)	16	0.05	0.00	16	0.00
can61 (61, 0.47)	46	0.82	2.11	46	0.11
can62 (62,0.11)	56	0.15	0.50	56	0.11
can73 (73, 0.25)	53	21.82	8.27	53	0.16
can96 (96, 0.20)	72	1131.60	154.92	72	0.44
curtis54 (54, 0.24)	46	0.20	0.63	46	0.07
dwt66 (66, 0.12)	62	0.06	0.31	62	0.10
dwt72 (72,0.07)	68	0.07	0.50	68	0.23
dwt87 (87, 0.19)	77	0.90	3.03	77	0.39
dwt_59 (59, 0.15)	51	0.30	0.97	51	0.10
fidap005 (27, 0.67)	18	0.04	0.11	18	0.01
fidapm05 (42, 0.61)	30	0.10	0.42	30	0.02
ibm32 (32, 0.36)	24	0.07	0.31	24	0.01
impcol_b (59, 0.19)	49	0.59	1.53	49	0.08
pores_1 (30, 0.41)	22	0.09	0.20	22	0.01
steam3 (80, 0.23)	72	0.81	1.83	72	0.20
west0067 (67, 0.19)	56	0.51	1.58	56	0.13
will57 (57, 0.19)	53	0.05	0.53	53	0.08
Average	49.63	48.46	7.87	49.63	0.13

Table 2: Results on the MM-II instances.

Instance	Opt.	SB Time	BM Time	RVNS	Time
arc130 (130, 0.93)	88	137.42	99.33	88	0.49
ash331 (104, 0.06)	97	0.82	2.81	97	0.91
bcspwr03 (118, 0.08)	112	0.38	2.42	112	1.38
bcsstk04 (132, 0.68)	84	80.01	26.80	84	1.21
can_144 (144, 0.16)	126	443.49	31.25	126	1.51
gre_115 (115, 0.09)	95	28.77	82.81	95	1.31
L125.ash608 (125, 0.05)	118	0.50	2.38	118	1.75
L125.bcsstk05 (125, 0.35)	101	54.29	30.81	101	1.27
L125.can_161 (125, 0.16)	97	1800.00	580.80	97	1.44
L125.can_187 (125, 0.13)	111	23.22	16.98	111	0.99
L125.dwt_162 (125, 0.12)	116	1.15	8.36	116	1.64
L125.dwt_193 (125, 0.38)	95	73.74	29.72	95	0.99
L125.fs_183_1 (125, 0.44)	98	36.21	35.02	97	0.98
L125.gre_185 (125, 0.15)	104	31.40	30.58	104	1.69
L125.lop163 (125, 0.16)	109	33.22	16.03	109	1.25
L125.west0167 (125, 0.06)	121	0.22	1.91	121	1.70
L125.will199 (125, 0.05)	119	0.35	2.56	119	1.17
lund_a (147, 0.26)	118	155.29	45.89	118	2.60
rw136 (136, 0.07)	121	57.57	28.88	121	2.36
west0132 (132, 0.06)	126	0.42	4.30	126	1.89
Average	110.45	147.92	53.98	107.75	1.43

Table 3: Results on the MM-HD instances.

Instance	Opt.	SB Time	BM Time	RVNS	Time
L100.cavity01 (100, 0.37)	85	3.52	10.66	85	0.57
L100.e05r0000 (100, 0.64)	70	14.94	7.94	70	0.38
L100.fidap001 (100, 0.68)	64	9.82	5.14	64	0.37
L100.fidap002 (100, 0.82)	66	2.38	6.23	66	0.31
L100.fidap021 (100, 0.41)	85	2.64	5.47	85	0.55
L100.fidap022 (100, 0.68)	62	30.62	13.99	62	0.39
L100.fidap025 (100, 0.41)	82	5.41	13.58	82	0.53
L100.fidap027 (100, 0.81)	69	4.15	23.44	69	0.35
L100.fidapm02 (100, 0.62)	69	6.35	25.13	69	0.43
L100.rbs480a (100, 0.52)	73	11.91	13.95	73	0.37
L100.steam2 (100, 0.36)	76	30.98	8.87	76	0.47
L100.wm1 (100, 0.60)	74	11.65	27.78	72	0.42
L100.wm2 (100, 0.61)	76	7.37	18.49	75	0.42
L100.wm3 (100, 0.59)	77	4.79	19.66	76	0.47
L120.cavity01 (120, 0.42)	99	16.57	16.38	99	0.98
L120.e05r0000 (120, 0.59)	90	7.51	9.83	90	0.76
L120.fidap001 (120, 0.63)	82	22.56	37.53	82	0.80
L120.fidap002 (120, 0.82)	68	37.86	10.45	68	0.55
L120.fidap021 (120, 0.43)	98	17.36	16.77	98	0.93
L120.fidap022 (120, 0.60)	84	94.06	30.78	84	0.87
L120.fidap025 (120, 0.39)	102	9.27	9.61	102	1.02
L120.fidap027 (120, 0.85)	83	9.07	50.72	83	0.58
L120.fidapm02 (120, 0.65)	86	12.29	42.95	86	0.81
L120.rbs480a (120, 0.46)	88	98.84	21.28	88	0.89
L120.wm2 (120, 0.47)	98	19.27	18.78	94	1.01
L80.cavity01 (80, 0.38)	65	2.22	4.58	65	0.25
L80.e05r0000 (80,0.68)	60	0.96	7.94	60	0.20
L80.fidap001 (80,0.72)	54	0.98	3.14	54	0.19
L80.fidap002 (80, 0.77)	53	1.71	3.33	53	0.16
L80.fidap021 (80,0.43)	65	2.43	4.03	65	0.23
L80.fidap022 (80,0.76)	41	10.05	2.17	41	0.11
L80.fidap025 (80,0.38)	68	1.14	2.63	68	0.32
L80.fidap027 (80,0.80)	56	1.18	4.47	56	0.18
L80.fidapm02 (80,0.65)	53	1.97	3.28	53	0.19
L80.rbs480a (80,0.58)	62	0.97	3.50	62	0.17
L80.steam2 (80,0.40)	61	3.15	4.22	61	0.25
L80.wm1 (80,0.57)	59	11.13	8.64	57	0.20
L80.wm2 (80,0.58)	61	2.89	7.36	61	0.21
L80.wm3 (80,0.55)	63	3.81	8.41	62	0.22
Average	72.49	13.74	13.67	72.21	0.46

Table 4: Results on the DIMACS instances.

Instance	Opt.	SB Time	BM Time	RVNS	Time
david (87, 0.11)	81	0.19	0.98	81	0.55
DSJC125.1 (125, 0.09)	91	1800.00	9783.08	90	0.63
DSJC125.5 (125, 0.50)	74	1800.00	8.61	74	0.62
DSJC125.9 (125, 0.90)	22	794.36	0.70	22	0.33
games120 (20, 0.09)	102	429.02	121.05	102	1.31
miles1000 (128, 0.40)	110	13.62	19.06	110	1.81
miles500 (128, 0.14)	119	2.11	7.16	119	1.96
miles750 (128, 0.26)	113	9.83	62.80	113	1.59
myciel3 (11, 0.36)	8	0.00	0.00	8	0.00
myciel4 (23, 0.28)	17	0.03	0.11	17	0.01
myciel5 (47, 0.22)	37	0.28	0.83	37	0.07
myciel6 (95, 0.17)	76	5.14	12.09	76	0.77
myciel7 (191, 0.13)	156	160.59	880.88	156	9.85
queen10_10 (100, 0.30)	67	1800.00	105.77	67	0.38
queen11_11(121, 0.27)	81	1800.00	456.78	81	0.65
queen12_12 (144, 0.25)	97	1800.00	1245.08	97	1.16
queen6_6 (36, 0.46)	21	1.42	0.02	21	0.02
queen7_7 (49, 0.40)	31	7.78	0.06	31	0.04
queen8_12 (96, 0.30)	65	1800.00	77.30	65	0.36
queen8_8 (64, 0.36)	43	42.44	0.03	43	0.11
queen9_9 (81, 0.33)	55	1067.50	14.38	55	0.22
Average	69.80	634.97	609.37	69.76	1.07

6. CONCLUSION

In this paper, we propose a Reduced Variable Neighborhood Search (RVNS) method to deal with the Vertex Separator (VS) problem. We present a very effective constructive procedure which allows the method to find very promising regions in the search space. In order to diversify the search, the RVNS is started from different initial solutions. In addition, we also present a shake procedure which balances intensification and diversification. We provide an extensive experimental comparison with the best previous method in the state of the art over a set of 104 instances. Experimental results show that the proposed algorithm algorithm find the optimal solution in most of the cases consuming really short computing times.

ACKNOWLEDGEMENT

This research has been partially supported by the Spanish Ministry of "Economía y Competitividad", grants ref. TIN2009-07516 and TIN2012-35632, and the Government of the Community of Madrid, grant ref S2009/TIC-1542.

REFERENCES

[1] Lipton, R.J. & Tarjan, R.J. (1979). A Separator Theorem for Planar Graphs. SIAM Journal on Applied Math, 36:177–189.

[2] Lipton, R.J. & Tarjan,R.E. (1980). Applications of a planar separator theorem. SIAM J. Comput., 9:615-–627.

[3] Leiserson, C. (1980). Area-efficient graph layouts (for VLSI). 21th Annual Symposium on Foundations of Computer Science. IEEE Computer Soc., Los Alamitos, CA, 270—280.

[4] Leighton F.T. (1983). Complexity Issues in VLSI: Optimal Layout for the Shuffle-Exchange Graph and Other Networks. MIT Press, Cambridge, MA.

[5] Bhatt,S.N. & Leighton,F.T.(1984). A framework for solving VLSI graph layout problems. J. Computer System Sci., 28:300—343.

[6] Bui,T.N. & Jones,C.(1992). Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett., 42:153—159.

[7] Mladenovic, N., Hansen P.,(1997). Variable neighborhood search. Computers and Operations Research 24 (11): 1097-–1100.

[8] Leighton, T. & Rao, S. (1999) Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM. 46:787-–832.

[9] Díaz, J., Petit, J. & Serna,M. (2002) A survey of graph layout problems. Journal ACM Computing Surveys, 34(3):313–356.

[10] Hansen, P. & Mladenović, N. (2003). Variable Neighborhood Search. International Series in Operations Research & Management Science, 57:145–184.

[11] Feige, U. & Kogan,S. (2004) Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report MCS04-04, Department of Computer Science and Applied Math., The Weizmann Institute of Science.

[12] Arora, S., Rao, S. & Vazirani,U. (2004).Expander flows, geometric embeddings, and graph partitionings. 36th Annual Symposium on the Theory of Computing, 222—231.

[13] de Souza, C. & Balas, E. (2005) The vertex separator problem: algorithms and computations. Mathematical Programming, 103:609-–631.

[14] Balas, E. & de Souza, C. (2005). The vertex separator problem: a polyhedral investigation. Mathematical Programming, 103:583–608.

[15] Feige, U., Hajiaghayi, M. & Lee, J.R. (2005). Improved approximation algorithms for minimum-weight vertex separators. Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, 563–572.

[16] Montemayor. A.S., Duarte A., Pantrigo J.J., Cabido R. (2005). High-Performance VNS for the Max-Cut Problem Using Commodity Graphics Hardware. Mini-Euro Conference on VNS (MECVNS 05), Tenerife (Spain), 1–11.

[17] Hansen, P., Mladenovic, N., Perez, J.A.M. (2010). Variable neighbourhood search: methods and applications. Annals of Operations Research 175:367-–407.

[18] Didi Biha, M. & Meurs, MJ. (2011). An exact algorithm for solving the vertex separator problem. Journal of Global Optimization, 49:425–434.

[**19**] Lozano, M., Duarte, A., Gortázar, F. & Martí R. (2012). Variable Neighborhood Search with Ejection Chains for the Antibandwidth Problem. Journal of Heuristics, 18:919–938.

[**20**] Duarte, A., Escudero L.F., Martí R., Mladenovic N., Pantrigo J.J., Sánchez-Oro, J. (2012). Variable Neighborhood Search for the Vertex Separation Problem. Computers and Operations Research, 39(12):3247—3255.

[**21**] Pardo, E.G., Mladenovic, N., Pantrigo, J.J. & Duarte, A. (2013). Variable Formulation Search for the Cutwidth Minimization Problem. Applied Soft Computing, 13:2242–2252.

[**22**] Sánchez-Oro, J., Pantrigo J.J., Duarte A. (2013). Balancing intensification and diversification strategies in VNS. An application to the Vertex Separation Problem. Technical Report. Dept. Ciencias de la Computación. Universidad Rey Juan Carlos.