Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Discrete Mathematics 39 (2012) 59–66

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

An experimental comparison of Variable Neighborhood Search variants for the minimization of the vertex-cut in layout problems

Jesús Sánchez-Oro[1] Abraham Duarte[1,2]

Dept. Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain

Abstract

Variable Neighborhood Search (VNS) is a metaheuristic for solving optimization problems based on a systematic change of neighborhoods. In recent years, a large variety of VNS strategies have been proposed. However, we have only found limited experimental comparisons among different VNS variants. This paper reviews three VNS strategies for finding near-optimal solutions for vertex-cut minimization problems. Specifically, we consider the min-max variant (Vertex Separation Problem) and the min-sum variant (SumCut Minimization Problem). We also present an preliminary computational comparison of the methods on previously reported instances.

Keywords: VNS, Profile, Vertex Separation, metaheuristic

[1] This research has been partially supported by the Spanish government, grants TIN2009-07516 and TIN2012-35632, and the Government of the Community of Madrid, grant S2009/TIC-1542

[2] Email:`abraham.duarte@urjc.es`

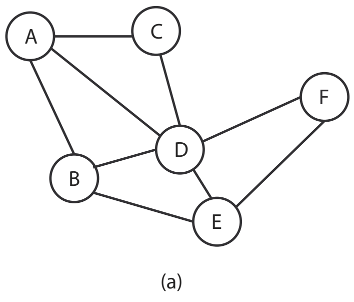
doi:10.1016/j.endm.2012.10.009

1 Introduction

Let $G(V, E)$ be an undirected graph where V and E are the sets of vertices and edges, respectively. A linear layout φ of the vertices of G is a bijection $\varphi : V \rightarrow \{1, 2, ..., n\}$ in which each vertex receives a unique and different integer between 1 and n. For vertex u, let $\varphi(u)$ denote its position or label in layout φ. Let $L(p, \varphi, G)$ be the set of vertices in V with a position in the layout φ lower than or equal to position p. Symmetrically, let $R(p, \varphi, G)$ be the set of vertices with a position in the layout φ larger than position p. In mathematical terms, $L(p, \varphi, G) = \{v \in V : \varphi(v) \leq p\}$ and $R(p, \varphi, G) = \{v \in V : \varphi(v) > p\}$.

Layouts are usually represented in a straight line, where the vertex (say for instance, u) in position 1 comes first (i.e., $\varphi(u) = 1$). Then, $L(p, \varphi, G)$ can be simply called the set of left vertices with respect to position p. Symmetrically, $R(p, \varphi, G)$ is the set of right vertices w.r.t. p.

The vertex-cut at position p of layout φ, denoted as $Cut(p, \varphi, G)$ is defined as the number of vertices in $L(p, \varphi, G)$ with one or more adjacent vertices in $R(p, \varphi, G)$. Then, $Cut(p, \varphi, G) = |\{u \in L(p, \varphi, G) : \exists v \in R(p, \varphi, G) \cap N(u)\}|$, where $N(u) = \{v \in V : (u, v) \in E\}$. Figure 1.a shows an example of an undirected graph, G, with six vertices and nine edges. Figure 1.b shows an example of a layout φ. For instance, the vertex-cut value in position $p = 4$ is $Cut(4, \varphi, G) = 2$ because vertices B and D with $\varphi(\text{B}) = 2$ and $\varphi(\text{D}) = 4$ (whose position is lower than or equal to 4) have an adjacent vertex in a position larger than 4.



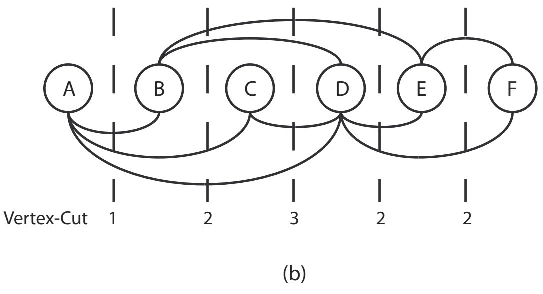


Fig. 1. Vertex-cut computation of the layout φ of graph G

We consider two optimization problems based on the minimization of the vertex-cut. In particular, in this paper we study the min-max variant, referred to as Vertex Separation Problem [3], and the min-sum variant, named SumCut Minimization Problem [7]. Given a layout φ of a graph G, the objective function of the Vertex Separation Problem is computed as the maximum of the vertex-cuts among all positions in layout φ. In mathematical terms,

$VS(\varphi, G) = \max_{1 \leq p \leq n} Cut(p, \varphi, G)$. The Vertex Separation Problem (VSP) consists of finding a layout, say φ^*, that minimizes the VS over all possible layouts. Attending to the layout depicted in Figure 1.b, the $VS(\varphi, G) = 3$.

Similarly, the SumCut objective function is computed as the sum of vertex-cuts of each position. In mathematical terms, $SC(\varphi, G) = \sum_{i=1}^{n} Cut(i, \varphi, G)$. The SumCut Minimization Problem (SCP) then consists of minimizing the value of SC over all possible layouts. Considering the example shown in Figure 1.b, $SC(\varphi, G) = 1 + 2 + 3 + 2 + 2 = 10$

These two problems are NP-complete (see [6] and [2]). Practical applications of the Vertex Separation and the SumCut problems can be found in [1] and [5].

2 Variable Neighborhood Search

Variable Neighborhood Search (VNS) [4] is a metaheuristic aimed at solving optimization problems. It is based on a systematic change of neighborhood combined with a local search. VNS variants are generally simple algorithms and require few search parameters (compared with other metaheuristics). Therefore, VNS emerges as good alternative to solve a wide range of optimization problems. From an algorithmic perspective, VNS consists of three main strategies: local search, shaking, and neighborhood change. VNS needs an initial solution to start, which is frequently constructed at random. In this paper, we use the constructive procedures described in [3] (VSP) and in [7] (SCP).

2.1 Local search procedures

The first local search (LS1) is based on interchanges. An $interchange(i, j)$ is the exchange of positions of vertices v, u that are currently in positions i and j, respectively. That is, $\varphi(i) = v$ and $\varphi(j) = u$. It starts by sorting the vertices according to their contribution to the objective function in descending order. This is based on the idea that moving the vertices with the largest contribution would reduce the objective function. Then, the method evaluates all possible interchanges for each vertex, finally performing the better one. The local search ends when no improvement is found.

The second local search (LS2) is based on insertions. An $insert(i, j)$ is the movement of vertex $v = \varphi(i)$ to position j. After the move, v precedes u if $i > j$ and v follows u if $i < j$. LS2 starts inserting each vertex in all the positions, placing it in the best one. LS2 ends when, after examining all the vertices, no improvements is found.

The third local search (LS3) is based on interchanges of a node and its associated adjacent vertices. The local search selects a vertex v at random and then starts moving it by interchanges to its best position in the layout. Then, LS3 do the same with the adjacent vertices of v, placing them in their best positions. The idea of moving the adjacent vertices is based on the hypothesis that they are the ones that would be affected by the move of v, and it may eventually ends in a better solution. If the movement of a vertex does not improve the objective function, LS3 discard it for future moves (to save CPU time). This local search ends when there are no available vertices to be moved.

2.2 Basic Variable Neighborhood Search

Figure 2.a shows the pseudocode of the Basic Variable Neighborhood Search (BVNS). For both VSP and SCP we consider LS2 as local search and the shaking procedure is implemented as a sequence of k consecutive insertions, as recommended in [3] and [7], respectively. In both problems, the neighborhood change strategy basically consists of increasing the value of k (non improving move) or setting it to 1 (improving move). The procedure ends when k reaches a pre-specified value (k_{max}).

2.3 Reduced Variable Neighborhood Search

Reduced Variable Neighborhood Search (RVNS) is an appropriate algorithm when we face large instances (the local search can be very time consuming). As it can be seen in Figure 2.b, this strategy does not consider a local search within the VNS framework. In our study, for the neighborhood change, the termination criterion, and shaking strategies, we consider the same configuration aforementioned.

2.4 Variable Neighborhood Descent

Variable Neighborhood Descent (VND) changes the neighborhood definition in order to obtain better solutions in the shaking step. The new neighborhood is defined by different local search methods. Specifically, the method starts with k_{max} different local searches. Then, in neighborhood k, the shaking step obtains a local optimum for local search k. The VND proposed in this work uses the three local searches (LS1, LS2 and LS3) described in Section 2.1. Figure 2.c illustrates the pseudocode of this algorithm.

```
BVNS($x$, $k_{max}$, $t_{max}$)
while ($t < t_{max}$)
  $k = 1$
  while ($k < k_{max}$)
    $x' \leftarrow$ Shake($x$,$k$)
    $x'' \leftarrow$ LocalSearch($x'$)
    NeighChange($x$,$x''$,$k$)
  endwhile
  $t \leftarrow$ Time()
endwhile
```

```
RVNS($x$, $k_{max}$, $t_{max}$)
while ($t < t_{max}$)
  $k = 1$
  while ($k < k_{max}$)
    $x' \leftarrow$ Shake($x$,$k$)
    NeighChange($x$,$x'$,$k$)
  endwhile
  $t \leftarrow$ Time()
endwhile
```

```
VND(x, $k_{max}$, $t_{max}$)
while ($t < t_{max}$)
  $k = 1$
  while ($k < k_{max}$)
    $x' \leftarrow$ LSSelection($x$, $k$)
    NeighChange($x$,$x'$,$k$)
  endwhile
  $t \leftarrow$ Time()
endwhile
```

Fig. 2. Pseudocode for BVNS, RVNS, and VND

3 Computational experience

This section reports the computational experiments that we performed to compare the previously described VNS variants for solving both, VSP and SCP problems. All procedures were implemented in Java SE 6 and all the experiments were conducted on an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB RAM. We derived 62 instances from the Harwell-Boeing Sparse Matrix Collection, where the number of vertices and edges range from 24 to 960 and from 34 to 3721, respectively. This set was previously used in [3] and [7].

In the first experiment we determine the best value of k_{max} for BVNS. In particular, we consider $k_{max} \in \{0.05n, 0.10n, 0.25n, 0.50n\}$ for both problems (VSP and SCP), where $n = |V|$. In order to have a fair comparison we limit the CPU time to 60 seconds for all experiments. Table 1 reports the average objective function (FO), the average deviation from the best known solution (Dev %) and the number of instances in which the algorithm matches the best solution (# Best).

Results reported in Table 1 clearly show that the best value for the VSP is $k_{max} = 0.05$, achieving the lowest average deviation, and matching the largest number of best values. In the case of the SCP, the BVNS with $k_{max} = 0.25$ clearly outperforms the other variants.

The next experiment is devoted to study the effect of k_{max} on RVNS, considering again VSP and SCP. We use the same values aforementioned for k_{max}. Table 2 shows the experimental behavior of the 4 RVNS variants. As in the previous experiment, the best result for the VSP and SCP is obtained when k_{max} is 0.05 and 0.10, respectively. It is important to remark that, the differences among the for RVNS variants are smaller than in BVNS variants.

In the next experiment, we investigate the effect of the order of the local

	VSP			SCP		
k_{max}	FO	Dev (%)	Best	FO	Dev (%)	# Best
0.05	24.66	0.16	61	8061.05	1.29	44
0.10	24.76	0.58	58	8052.50	0.98	41
0.25	24.81	1.34	56	8046.74	0.63	51
0.50	24.79	0.80	57	8054.19	0.91	50

Table 1
k_{max} selection for BVNS

	VSP			SCP		
k_{max}	FO	Dev (%)	Best	FO	Dev (%)	# Best
0.05	26.73	0.00	62	8432.87	0.11	60
0.10	26.74	0.20	61	8433.16	0.08	54
0.25	26.74	0.20	61	8433.32	0.13	53
0.50	26.74	0.20	61	8433.60	0.17	51

Table 2
k_{max} selection for RVNS

search procedures within the VND template. We consider the three local search methods described in Section 2.1: LS1, LS2, and LS3. Table 3 shows the results of the associated 6 variants. In the VSP we can observe that LS1-LS3-LS2 and LS1-LS2-LS3 obtain the same results. We finally choose, LS1-LS3-LS2 since it performance is slightly better. In the particular case of the SCP the best variant is LS1-LS2-LS3, which slightly outperforms the other ones.

Once we have identified the best parameter for each VNS variant, we compare all of them in order to determine the best algorithm for each problem. We consider two time horizons: 60 seconds and 600 seconds. Tables 4 and 5 show that BVNS is the best variant in both problems and both time horizons. The RVNS method systematically presents worse performance than BVNS over the whole set of instances. These results can be partially explained since RVNS is suitable when we are facing very large instances (the largest instance in our experimentation has around 900 vertices) and when the local search method is computationally expensive (our local search strategies are extremely efficient).

	VSP			SCP		
k_{max}	FO	Dev (%)	Best	FO	Dev (%)	# Best
LS1 - LS3 - LS2	30.55	17.61	31	11140.37	25.73	6
LS1 - LS2 - LS3	26.85	0.91	57	8393.53	0.70	37
LS3 - LS1 - LS2	30.73	19.19	17	10782.00	24.51	4
LS3 - LS2 - LS1	30.39	17.89	26	10771.10	25.06	3
LS2 - LS1 - LS3	26.74	0.43	58	8432.37	0.60	36
LS2 - LS3 - LS1	26.74	0.43	58	8432.37	0.60	36

Table 3
Local search order in VND

Finally, the VND seems to be the method with the worst results. However, it could be partially explained since one of the proposed local search strategies clearly outperforms the other two (LS1 in the SCP and LS2 in the VSP). Therefore, the change of neighborhood (by using other local search) does not really improve the results.

	VSP			SCP		
k_{max}	FO	Dev (%)	Best	FO	Dev (%)	# Best
BVNS	24.66	0.00	62	8046.74	0.01	60
RVNS	26.73	10.94	29	8432.87	8.43	6
VND	26.74	11.63	28	8393.53	8.73	5

Table 4
Final comparison with a time horizon of 60 seconds

	VSP			SCP		
k_{max}	FO	Dev (%)	Best	FO	Dev (%)	# Best
BVNS	23.26	0.00	62	7615.50	0.00	62
RVNS	26.69	15.73	27	8428.68	12.65	3
VND	26.74	17.01	25	8393.53	13.12	3

Table 5
Final comparison with a time horizon of 600 seconds

4 Conclusions

A experimental comparison of three VNS variants for two vertex-cut minimization problems has been presented. In particular, we have considered the Vertex Separation Problem and the SumCut Minimization Problem. Experiments with 62 instances have been performed in order to compare this three variants over two hard optimization problems. Our experimentation reveals that the Basic VNS strategy is the best option when considering short and long time horizons. However we believe that this experimental study should be enlarged in order to draw more robust conclusions.

References

[1] Botafogo, R. A., *Cluster analysis for hypertext systems*, Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval (1993), 116–125.

[2] Díaz, J., A.M. Gibbons, M. S. Paterson, and J. Torán, *The Minsumcut problem.* Algorithms and Data Structures **519** (1991), 65–79.

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

[4] Hansen, P., N. Mladenović, J. Brimberg, and J. A. Moreno Pérez, *Variable neighbourhood search.* Handbook of Metaheuristics, (second edition) Springer (2010) 61–86.

[5] Leiserson C. E., *Area-efficient graph layouts (for VLSI)*, Proceedings of IEEE symposium on foundations of computer science (1980), 270–281.

[6] Lengauer T, *Black-white pebbles and graph separation*, Acta Informatica **16** (1981), 465–475.

[7] Sánchez-Oro, J., and A. Duarte, *GRASP with Path Relinking for the SumCut Problem*, International Journal of Combinatorial Optimization Problems and Informatics **3** (2011), 3–11.