Breakout local search for the vertex separator problem

Una Benlic and Jin-Kao Hao
LERIA, Université d'Angers
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
{benlic,hao}@info.univ-angers.fr

Abstract

In this paper, we propose the first heuristic approach for the vertex separator problem (VSP), based on Breakout Local Search (BLS). BLS is a recent meta-heuristic that follows the general framework of the popular Iterated Local Search (ILS) with a particular focus on the perturbation strategy. Based on some relevant information on search history, it tries to introduce the most suitable degree of diversification by determining adaptively the number and type of moves for the next perturbation phase. The proposed heuristic is highly competitive with the exact state-of-art approaches from the literature on the current VSP benchmark. Moreover, we present for the first time computational results for a set of large graphs with up to 3000 vertices, which constitutes a new challenging benchmark for VSP approaches.

Keywords: vertex separator problem, breakout local search, adaptive diversification mechanism, meta-heuristic.

1 Introduction

A vertex separator in a graph is a set of vertices whose removal disconnects the graph in at least two non-empty connected components. More formally, given a connected undirected graph $G = (V, E)$, a cost c_i associated to each vertex $i \in V$, and an integer $1 \leq b \leq |V|$, the vertex separator problem (VSP) is to find a partition of V into disjoint subsets A, B, C with A and B non-empty, such that (i) there is no edge $(i, j) \in E$ such that $i \in A$, $j \in B$, (ii) $max\{|A|, |B|\} \leq b$ and (iii) $\sum\{c_j : j \in C\}$ is minimized. A and B are called shores of the separator C. A separator C is a legal (feasible) solution if it satisfies the problem constraints (i) and (ii), and is termed optimal if (i), (ii) and (iii) are met. VSP is an NP-hard constraint problem [Bui and Jones, 1992; Fukuyama, 2006] which can be used to test AI heuristics and search methods. It is useful for many graph algorithms and has a number of applications: VLSI design, communication networks, bioinformatics (see [Balas and de Souza, 2005a] for a survey of VSP applications).

Several exact algorithms have been proposed for solving VSP [Balas and de Souza, 2005a; 2005b; Biha and Meurs, 2011; Cavalcante and de Souza, 2011]. These VSP algorithms are able to find optimal results in a reasonable computing time (within 3 hours) for instances with up to 150 vertices, and may fail to solve larger instances.

On the other hand, heuristic and meta-heuristic methods, which have shown to be very useful for various NP-hard combinatorial optimization problems, have not been considered up until now for VSP. Although such methods have no formal performance guarantee, they have shown to be able to provide solutions of acceptable quality with reasonable computing efforts even for very large instances.

In this paper, we present the first heuristic algorithm for VSP, based on Breakout Local Search (BLS). BLS is a recent meta-heuristic which has been successfully applied to several classic combinatorial optimization problems including minimum sum coloring problem [Benlic and Hao, 2012], maximum clique [Benlic and Hao, 2013a], quadratic assignment [Benlic and Hao, 2013b] and maximum cut [Benlic and Hao, 2013c]. Based on the framework of Iterated Local Search (ILS) [Lourenco *et al.*, 2003], BLS combines local search (i.e., the steepest descent) with a dedicated and adaptive perturbation mechanism. For each perturbation phase, BLS tries to establish the most suitable degree of diversification by determining dynamically the number of perturbation moves (i.e., the jump magnitude) and by adaptively choosing between several types of perturbation moves of different intensities. This is achieved through the use of information from dedicated memory structures. In contrast with the BLS approaches for the previously considered problems, the proposed BLS for VSP incorporates a more advanced reactive technique, based on the occurrences of cycles, to determine the jump magnitude for the next perturbation phase. Moreover, it employs a novel mechanism to adaptively choose between two types of perturbation moves.

Tested on the set of 104 VSP benchmark instances from the literature (with less than 200 vertices), the proposed BLS easily attains an optimal solution for all these VSP instances. Since heuristic methods are mainly conceived to tackle large instances of NP-hard problems, we further present for the first time computational results for 54 new and large graphs with up to 3000 vertices.

2 Breakout Local Search (BLS) for VSP

2.1 General framework of BLS

BLS is a recent general stochastic local search method which follows the basic scheme of the iterated local search (ILS). Its basic idea is to use a descent-based local search procedure to intensify the search in a given search space region, and to apply effective perturbations to move to a new search region once a local optimum is attained. BLS has a particular focus on the importance of the perturbation phase, and uses an adaptive and multi-typed perturbation mechanism to introduce a suitable degree of diversification at a certain stage of search. A BLS algorithm requires four procedures to be specified: $GenerateInitialSolution$ generates a starting point for the search; $DescentBasedSearch$ is the descent/ascent local search procedure which searches a defined neighborhood for a solution which is better than the current solution, and terminates if such a solution is not found; $DetermineJumpMagnitude$ determines the number L of perturbation moves ("jump magnitude"); $DeterminePerturbationType$ determines the type T of perturbation moves among two or several alternatives of different intensities. Once the number L and the type T of perturbation moves are selected, BLS calls the $Perturb$ procedure which applies L moves of type T to the current local optimum (we say that the solution is perturbed). This perturbed solution becomes the new starting point for the next round of the descent-based local search. Alg. 1 gives an algorithmic scheme for BLS. The history component in $DetermineJumpMagnitude$, $DeterminePerturbationType$ and $Perturb$ indicates that some relevant information on the search history is used to influence the decisions made in these procedures.

We next provide details on the four component procedures of our BLS algorithm for VSP.

Algorithm 1 BLS general framework

1: $p' \leftarrow$ GenerateInitialSolution
2: $L \leftarrow L_0$ /*Initialize the number L of perturbation moves */
3: **while** stopping condition not reached **do**
4: $\quad p \leftarrow DescentBasedSearch(p')$
5: $\quad L \leftarrow DetermineJumpMagnitude(L, p, history)$
6: $\quad T \leftarrow DeterminePerturbationType(p, history)$
7: $\quad p' \leftarrow Perturb(L, T, p, history)$
8: **end while**

2.2 Initial solution

To generate an initial solution, we use a simple heuristic procedure which performs in two steps. First, it randomly places vertices of V to either A or B such that $1 \leq |A|, |B| \leq b$ (constraint (ii)), without caring about the problem constraint (i) which requires that there is no edge $(i, j) \in E$ such that $i \in A$ and $j \in B$. Second, constraint (i) is fulfilled by moving to the separator C either a vertex $i \in A$ or $j \in B$ if $(i, j) \in E$. However, the resulting initial solution may still be illegal since A or B can become empty in the second step, even though $|C| < |V|$. This constraint is met during the first iterations of BLS since the next move always involves displacing a vertex of C to an empty shore subset, i.e., to $S \in \{A, B\}$, $|S| = 0$.

2.3 Neighborhood and local search

To move from one solution to another in the search space, BLS employs a move operator $Move(v, S)$ which selects a vertex v from the separator C ($v \in C$) and moves it to a shore subset $S \in \{A, B\}$. Afterwards, the resulting solution is repaired by moving to C the adjacent vertices $w \in O$, $(w, v) \in E$ from the opposite shore subset O (i.e, $O \neq S \neq C$). It is important to mention that a performed move never violates the problem constraint (ii) which requires that $1 \leq |A|, |B| \leq b$. Indeed, a vertex $v \in C$ is not considered for transfer to the shore subset S if $|S| = b$ or if all the vertices would have to be moved from the opposite shore subset O to C to repair the resulting solution (i.e., if $(|O| - |\{w : (v, w) \in E \text{ and } w \in O, v \in S\}|) = 0$). The neighborhood $N(p)$ of a solution $p = \{A, B, C\}$ can then be defined as the set of all the solutions which can be obtained with the move operator $Move(v, S)$ without violating the problem constraints.

The key concept related to the defined neighborhood is the move gain $g(v, S)$, which represents the change in the optimization objective. It expresses an estimate on how much a solution could be improved if a vertex $v \in C$ is moved to the shore subset S. Given a vertex $v \in C$, we compute gains $g(v, A)$ and $g(v, B)$ for moving v to shores A and B respectively. The selection of the vertex with the highest gain, as well as the updates needed after each move, are achieved efficiently with an adaptation of bucket sorting that has been extensively used to speed up the performance of graph partitioning algorithms [Fiduccia and Mattheyses, 1982; Benlic and Hao, 2011].

Each iteration of the descent-based local search phase consists in identifying the best legal move (v, S) and applying it to the current solution $p = \{A, B, C\}$ to obtain a new solution. This process is repeated until a local optimum is reached. After each move, the bucket sort structure is updated by recomputing gains for vertices affected by the move.

2.4 Adaptive diversification strategy

The perturbation mechanism plays a crucial role within BLS since the descent-based local search procedure alone cannot escape from a local optimum. The success of the described method depends crucially on two factors. First, it is important to determine the number L of perturbation moves ("jump magnitude") to be applied to change (i.e., perturb) the solution. Second, it is equally important to consider the type of perturbation moves to be applied. While conventional perturbations are often based on random moves, more focused perturbations using dedicated information could be more effective. The degree of diversification introduced by a perturbation mechanism depends both on the jump magnitude and the type of moves used for perturbation. If the diversification is too weak, the local search has greater chances to end up cycling between two or more locally optimal solutions, leading to search stagnation. On the other hand, a too strong diversification has the same effect as a random restart, which usually results in a low probability of finding better solutions in the

following descent-based local search phase. The proposed BLS for VSP employs directed and random perturbation operators, the former being based on history information maintained in a recency based tabu list [Glover, 1989]. These two types of perturbation introduce different degrees of diversification into the search. To determine the most suitable number and type of perturbation moves at a given stage of the search, the proposed BLS for VSP takes advantage of the information related to the occurrences of cycles. This information is based on a number of most recently visited locally optimal solutions stored in a hash table structure. We next provide some details on hashing VSP solutions and storing relevant information gained during the search. Afterwards, we describe in detail the proposed diversification mechanism, along with the perturbation types used, and explain how the information from the hash table memory is used to introduce a suitable amount of diversification at a certain search stage.

Hashing function and hash table structure

Each time a local optimum $p = \{A, B, C\}$ is attained after the descent-based local search phase (see Section 2.3), we store p in a hash table HT (if it is not already in HT), along with the iteration number at which p was last visited.

More precisely, we first compute an index h into the hash table, from which solution $p = \{A, B, C\}$ can be found, by mapping $p = \{A, B, C\}$ to an integer value using the hash function of the form [Woodruff and Zemel, 1993]

$$h = [\sum_{i \in C} z_i] \bmod [MAXHS + 1], \qquad (1)$$

where $MAXHS$ is the size of the hash table ($MAXHS = 100000$ in our experiments) and z is a precomputed vector of pseudo-random integers in the range $[1..131072]$.

If there is no solution recorded at HT_h, we store the local optimum p and its related information in HT_h. Otherwise, we compare p with the solution stored at HT_h. If these two local optima are identical, a cycle is encountered and the relevant information stored at HT_h (i.e., the iteration number at which p was previously visited) is used to determine the diversification degree required to break the current cycle. If p and HT_h are not identical, we keep passing to the next location until the identical solution is encountered or a free position is found to place p along with its related information.

Once the number of local optima stored in HT exceeds a given limit $MAXSLO$ ($MAXSLO = 500$ in our experiments), the least recently visited local optimum, as well as all the information related to this local optimum, are removed from HT to make place for the most recently visited local optimum. In order to determine in $O(1)$ this local optimum to be removed from HT, we maintain the indicator set data structure detailed in [Battiti and Protasi, 2001].

Determining jump magnitude

After a local optimum p is attained during the descent-based local search phase, BLS determines a suitable number of perturbation moves for the next perturbation phase. This procedure is given in Alg. 2.

BLS first calls a function $PreviousEncounter$ (line 1 of Alg. 2) which checks whether p is already in the hash table memory (see the last sub-section). If p is not in HT, the function $PreviousEncounter$ returns -1, and p is inserted into HT along with the descent-phase number $iter_{cur}$ at which p is visited. Otherwise, if p has previously been visited during the search (i.e., p is already in HT), the record in HT corresponding to the local optimum p is updated with the current descent-phase number $iter_{cur}$, and the function returns the descent-phase number $prev\_visit$ at which p has been encountered earlier. The information returned by the $PreviousEncounter$ function, stored in variable $prev\_visit$, is later used to determine the appropriate perturbation type for the current state of the search.

If a cycle is encountered (i.e., $prev\_visit \neq -1$), BLS increments the number of perturbation moves to increase slightly the degree of diversification (line 4 of Alg. 2). Otherwise, the number of perturbation moves is decreased if a cycle has not been detected for at least $wc \cdot \beta$ descent phases, where wc is the average number of descent phases between two consecutive cycles and β a coefficient (lines 5-7 of Alg. 2). Finally, we limit the number of perturbation moves to take values in the interval $[L_{MIN}, L_{MAX}]$ (lines 9-12 of Alg. 2).

Algorithm 2 DetermineJumpMagnitude(L, p, HT, lc, $iter_{cur}$)

Require: Local optimum p returned by $DescentBasedSearch$, current jump magnitude L and history information including the hash table HT of previously encountered local optima, the descent-phase number lc when the last cycle was encountered, and the total number of descent phases $iter_{cur}$

Ensure: Jump magnitude L for the next perturbation phase

/* Check whether p has previously been encountered (whether p is in HT) */
1: $prev\_visit \leftarrow PreviousEncounter(HT, p, iter_{cur})$
2: **if** $prev\_visit \neq -1$ **then**
3: $lc \leftarrow iter_{cur}$ /* Update the descent-phase number at which the last cycle is detected*/
4: $L \leftarrow L + 1$ /* Increment the jump magnitude*/
5: **else if** $(iter_{cur} - lc) > wc \cdot \beta$ **then**
6: /*A cycle has not been detected for at least $wc \cdot \beta$ descent phases; wc is the average number of descent phases between two consecutive cycles and β a coefficient*/
7: $L \leftarrow L - 1$ /* Decrement the jump magnitude */
8: **end if**
/* Limit L to take values no larger than L_{MAX} and no smaller than L_{MIN} */
9: **if** $L > L_{MAX}$ **then**
10: $L \leftarrow L_{MAX}$
11: **else if** $L < L_{MIN}$ **then**
12: $L \leftarrow L_{MIN}$
13: **end if**

Adaptive combination of two perturbation types

To perturb the current local optimum p, BLS adaptively chooses between *directed perturbation* and *random perturbation*.

The *directed perturbation* (DIRP) is based on the tabu search principles [Glover, 1989]. It uses a selection rule that favors the *legal* moves (see Section 2.3) which minimize the

degradation of the objective, under the constraint that these moves are not prohibited by the tabu list. Move prohibition is determined in the following way. Each time a vertex v is moved from its current shore subset $S \in \{A, B\}$ to C, it is forbidden to place it back to S for γ iterations (called tabu tenure), where γ takes a random value from a given range. The information for move prohibition is maintained in the matrix $H_{i,S}$, $i \in V$, $S \in \{A, B\}$ where the element (i, S) is equal to γ plus the iteration number when the vertex i was last moved from its shore subset S to the separator C. The tabu status of a move is neglected only if the move leads to a new solution better than the best solution found so far. The *directed perturbation* relies thus both on 1) history information which keeps track, for each move, of the last time (iteration) when it was performed and 2) the quality of the moves to be applied for perturbation in order not to deteriorate too much the perturbed solution.

The *random perturbation* (RNDP) performs random moves that do not violate the problem constraint (ii) on the resulting size of the shore subsets. In other words, it consists in randomly selecting a vertex $v \in C$ to be moved to a randomly chosen shore subset $S \in \{A, B\}$, under the constraint that $1 \leq |A|, |B| \leq b$ once the move is realized.

These two perturbation types do not introduce the same degree of diversification into the search. It is obvious that DIRP is more oriented towards search intensification than RNDP since DIRP also relies on the quality of moves in order not to degrade too much the resulting solution.

The proposed BLS considers the search history information from the hash table structure to determine the most appropriate perturbation type to be applied to the current local optimum p. Let $prev\_visit$ be the descent-phase number at which the current local optimum has been previously attained, the type of perturbation moves T is determined with the following relations:

$$e = \begin{cases} \alpha_{nc} + \frac{iter_{cur} - lc}{maxinc} & \text{if } prev\_visit = -1 \\ 1 - \alpha_c - \frac{iter_{cur} - prev\_visit}{\#lo\_in\_HT} & \text{otherwise} \end{cases} \quad (2)$$

and

$$T = \begin{cases} DIRP & \text{if } e \geq random(0, 0.01, 0.02, ..., 1) \\ RNDP & \text{otherwise} \end{cases} \quad (3)$$

where $iter_{cur}$ is the total number of performed descent phases, lc the number of the last descent-phase which led to a previous local optimum (see line 3 of Alg. 2), $maxinc$ the maximal number of consecutive descent phases without returning to a previous solution, $\#lo\_in\_HT$ the number of local optima stored in the hash table, and α_{nc} and α_c two positive coefficients that take a real value in the range $[0, 1]$.

The rationale of Eq. 2 is as follows. In the first case, when p has not been visited recently (i.e., p is not in HT and thus $prev\_visit = -1$), BLS determines the probability of applying the directed over the random perturbation moves depending on the difference $iter_{cur} - lc$, i.e., the number of consecutive descent phases elapsed without returning to a previous local optimum from HT. The larger the difference $iter_{cur} - lc$ is, the less the chance the search is stuck in a cycle. Therefore, the larger the $iter_{cur} - lc$, the higher the probability of using the directed perturbation over the random one. In the second case, when the search returned to the local optimum p from the hash table memory (i.e., when $prev\_visit \neq -1$), BLS determines the probability of using the directed perturbation depending on the difference $iter_{cur} - prev\_visit$ i.e., the number of descents elapsed before returning to p. The larger the difference $iter_{cur} - prev\_visit$ is, the higher the chances the search is stuck in a large cycle. Since large cycles are more difficult to break than small ones, Eq. 2 ensures that the larger the $iter_{cur} - prev\_visit$, the larger the probability of performing random perturbation moves which introduce more diversification into the search. BLS exhibits a bias in favor of the directed perturbation with coefficient α_{nc} when $prev\_visit = -1$, and is biased against the directed perturbation moves with α_c in case $prev\_visit \neq -1$. If e is greater than a random real number selected from the range $[0, 1]$ (with a step of 0.01), BLS applies L moves of the directed perturbation to the current solution p. Otherwise, it performs L moves of the random perturbation (see Eq. 3).

3 Experimental results

3.1 Experimental protocol and benchmark instances

Our BLS algorithm is programmed in C++ and compiled with GNU g++ under GNU/Linux running on an Intel Xeon E5440 with 2.83 GHz and 2 GB of RAM. Following the DIMACS machine benchmark[1], our machine requires 0.23, 1.42 and 5.42 CPU seconds for r300.5, r400.5, and r500.5 respectively. In our experiments, we set the minimum L_{MIN} and the maximum L_{MAX} number of perturbation moves to $0.05|C|$ and $0.25|C|$ respectively, and the coefficients α_{nc}, α_c, β to 0.6, 0.2 and 4 respectively. The tabu tenure γ, used for the directed perturbation, takes a random value in the range $[0.2|C|, 0.7|C|]$. These parameter values are determined in a preliminary experiment.

We evaluate extensively the performance of our BLS algorithm on the current VSP benchmark (104 instances in total)[2] which consists of: intersection graphs obtained from the coefficient matrices of linear equations (*MatrixMarket* (MM) instances classified into three types: MM-I, MM-II and MM-HD) and instances taken from the DIMACS challenge on graph coloring. These graphs are of different densities with $|V| \leq 191$. The optimal solution is known for all these 104 VSP instances. Since BLS is able to attain, with a 100% success rate, the optimal solution for the complete benchmark, we further use a set of 54 more challenging graphs generated by Helmberg and Rendl [Helmberg and Rendl, 2000]. This benchmark[3] consists of toroidal, planar, and random graph instances ranging from $|V| = 800$ to 3000. For our experiments on these graphs, all the vertices are given unit weights, and $b = \lfloor \frac{1.05|V|}{2} \rfloor$.

[1] ftp://dimacs.rutgers.edu in directory /pub/dsj/clique

[2] http://www.ic.unicamp.br/∼cid/Problem-instances/VSP.html#VSP

[3] http://www.optsicom.es/maxcut/#instances

3.2 Computational results on existing benchmark

We compare our BLS algorithm with the current state-of-art VSP approaches proposed in [Balas and de Souza, 2005b] and [Biha and Meurs, 2011], using the existing VSP benchmark from the literature.

The results of the reference approaches are taken from the corresponding papers. The results reported in [Balas and de Souza, 2005b] were obtained on a desktop PC equipped with a Pentium 4 processor, with 2.5 GHz and 2 GB of RAM. In [Biha and Meurs, 2011], experiments were performed on a Laptop Computer equipped with a Pentium M740 processor, with 1.73 GHz and 1 GB of RAM. The maximum run-time limit for BLS used in this experiments is 10 seconds. We use the time required to reach the optimum as the main criterion for this comparison. However, a completely fair comparison is impossible since we have a nondeterministic approach (i.e., our approach) on the one hand and the deterministic approaches on the other hand. In addition, different computing environments used constitute another major source of difficulty for a fair comparison. Since no sufficient information is available to benchmark the computers used in [Balas and de Souza, 2005b] and [Biha and Meurs, 2011], the reported times are only given for indicative purposes.

Table 1 summarizes the time requirements (in seconds) for BLS and the reference algorithms. We provide for each class of instances and each algorithm the average t_{avg}, the best t_{best} and the worst t_{worst} average time requirement over the given number of instances. For BLS, we take into account for each instance the worst case time needed to attain an optimal solution over 100 runs. On the other hand, we consider the best time among the five codes used in [Balas and de Souza, 2005b] to report results obtained by Balas and Souza. Moreover, we show for each approach the number of instances from a given class that were solved to optimality (row *# solved inst.*).

From Table 1, we observe that our BLS is highly competitive with the reference approaches. Indeed, for all the types of VSP instances, the average time t_{avg} required by BLS is less than 0.2 seconds, which is negligible compared to the time required by the reference methods. The worst time for BLS, shown in column t_{worst}, does not exceed 3.1 seconds. As the recent approach by Biha and Meurs, our algorithm is able to attain the optimal solutions for all the MM-I, MM-II, MM-HD and DIMACS instances with a 100% success rate. On the other hand, the method by Balas and Souza fails to reach the optimal solution for 7 out of the 104 instances within the CPU time limit of 30 minutes which is fixed by the authors.

3.3 Computational results on new benchmark

In this section, we present computational results of BLS on the set of 54 (new) large and challenging graphs (Table 2), which can serve as a reference for other new VSP approaches. To ease future comparisons, we make our results available at *http://www.info.univ-angers.fr/pub/hao/BLSVSP.html*. Another goal of this section is to show the interest of BLS's adaptive and multi-typed perturbation mechanism by comparing BLS with three versions of iterated local search (denoted as ILS-I, ILS-II and ILS-III). These three ILS algorithms are obtained with slight modifications of BLS. ILS-I is obtained by fixing the probability of applying directed versus random perturbation to 0.95. ILS-II is obtained by fixing both the number of perturbation moves ($L = 0.15|C|$) and the probability of applying directed versus random perturbation ($e = 0.95$). ILS-III is the most basic version of ILS which applies a fixed number ($L = 0.02|C|$) of random perturbation moves once a local optimum is attained by the descent phase. For these four algorithms (BLS, ILS-I, ILS-II and ILS-III), we use the default parameter setting given in Section 3.1. We run the four algorithms under the same computing conditions, i.e., 100 executions per instance with the time limit set to 1 hour.

The comparative results are provided in Table 2. The second column gives the best objective value obtained by BLS after fine-tuning its parameters. For each algorithm, we show the best and average result (column $best(avg)$) over 100 executions, and the average time in seconds required to attain the best result from $best(avg)$. The last row shows, for each algorithm, the number of instances for which the best-known objective value (column $best$) is attained. We observe that BLS attains the best-known result from column $best$ for 53 out of the 54 instances while ILS-I, ILS-II and ILS-III fail to reach the best-known result for 12, 11 and 47 instances respectively. As expected, the most basic algorithm ILS-III, which relies uniquely on random perturbations to diversify the search, shows the worst performance compared to the other three approaches. Moreover, we note that ILS-I outperforms ILS-II on 5 instances in terms of solution quality, and is outperformed on 2 instances. This additionally confirms the benefit of the adaptive combination of multiple perturbation types based on the search history. To see whether there exists significant performance difference in terms of solution quality among BLS and the three ILS algorithms, we apply the Friedman non-parametric statistical test followed by the Post-hoc test on the results from Table 2. This test shows that there is a significant performance difference among the compared algorithms with a p-value less than $2.2e^{-16}$. The Post-hoc analysis shows that BLS statistically outperforms ILS-III with a p-value of 0.000 while its dominance over ILS-I and ILS-II is less pronounced.

4 Conclusion

The presented BLS algorithm is the first meta-heuristic approach for the Vertex Separator Problem which is a NP-hard constraint combinatorial search problem. BLS combines a descent-based local search with an adaptive combination of multi-typed perturbations to ensure a desired balance between intensification and diversification of the search process. The evaluation of BLS on the whole set of 104 current VSP instances showed that BLS is able to attain, with a 100% success rate and very quickly (generally in less than one second), the optimal solution for the complete benchmark. This implies that the commonly used VSP instances are easy for our BLS algorithm. For this reason, we further presented computational results of BLS on 54 large and challenging graphs (with up to 3000 vertices). These results can serve as references for future VSP approaches.

Table 1: Comparison of BLS with two exact algorithms presented in [Balas and de Souza, 2005b] and [Biha and Meurs, 2011] on the current VSP benchmark (104 instances).

	BLS	Balas-Souza	Biha-Meurs		BLS	Balas-Souza	Biha-Meurs
MM-I instances				MM-II instances			
t_{avg}(s)	0.02	48.46	7.87	t_{avg}(s)	0.06	60.98	53.93
t_{best}(s)	0.00	0.02	0.00	t_{best}(s)	0.00	0.33	1.38
t_{worst}(s)	0.36	1131.60	154.92	t_{worst}(s)	0.09	443.49	580.80
# solved inst.	24/24	24/24	24/24	*# solved inst.*	20/20	19/20	20/20
MM-HD instances				DIMACS instances			
t_{avg}(s)	0.11	13.74	13.67	t_{avg}(s)	0.14	168.95	609.37
t_{best}(s)	0.00	0.96	2.17	t_{best}(s)	0.00	0.00	0.00
t_{worst}(s)	3.06	98.84	50.72	t_{worst}(s)	2.44	1067.50	9783.08
# solved inst.	39/39	39/39	39/39	*# solved inst.*	21/21	15/21	21/21

Table 2: Comparison of BLS with three variants of ILS on graphs generated by Helmberg and Rendl (54 instances).

		BLS		ILS-I		ILS-II		ILS-III	
Name	best	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)
G1	257	257(257)	7.2	257(257)	1.9	257(257)	2.2	258(258.98)	21.2
G2	257	257(257)	8.7	257(257)	2.9	257(257)	2.0	258(258.99)	6.4
G3	257	257(257)	66.3	257(257)	15.0	257(257)	13.0	258(258.99)	12.8
G4	363	363(363.17)	936.4	363(363.05)	854.1	363(363.02)	977.4	367(367.1)	1176.3
G5	257	257(257)	65.9	257(257)	15.6	257(257)	18.7	259(259)	11.2
G6	257	257(257)	8.5	257(257)	2.0	257(257)	2.1	259(259)	14.0
G7	257	257(257)	10.0	257(257)	2.6	257(257)	2.6	259(259)	10.0
G8	257	257(257)	69.6	257(257)	13.4	257(257)	15.9	258(258.99)	9.5
G9	257	257(257)	34	257(257)	7.9	257(257)	14.2	258(258.99)	12.9
G10	257	257(257)	70.7	257(257)	19.0	257(257)	22.2	258(258.99)	4.8
G11	16	16(16)	0.0	16(16)	0.0	16(16)	0.1	16(16)	0.0
G12	32	32(32)	0.0	32(32)	0.0	32(32)	0.0	32(32)	0.0
G13	45	45(45)	0.2	45(45)	6.4	45(45)	10.4	48(49.91)	1.3
G14	146	146(146.88)	768.7	146(146.24)	1061.2	146(146.19)	1006.1	179(181.98)	1160.4
G15	144	144(144.02)	856.7	144(144)	25.1	144(144)	39.8	178(181.33)	875.2
G16	144	144(144)	188.6	144(144)	10.4	144(144)	9.8	177(181.22)	335.0
G17	144	144(144)	401.1	144(144)	31.1	144(144)	31.1	178(180.47)	1719.7
G18	146	146(146.88)	623.5	146(146.2)	1174.0	146(146.33)	1045.0	179(182.23)	1556.9
G19	144	144(144.1)	1085.8	144(144)	20.3	144(144)	25.6	176(181.2)	1619.6
G20	144	144(144)	353.2	144(144)	12.0	144(144)	16.2	176(181.39)	416.9
G21	144	144(144)	552.5	144(144)	31.3	144(144)	44.2	176(180.4)	2445.3
G22	588	588(588.82)	657.7	588(588.83)	569.1	588(588.86)	411.7	617(618.57)	857.8
G23	590	590(590.97)	310.4	590(590.99)	220.8	590(590.98)	1105.9	618(618.85)	1023.0
G24	589	589(589.87)	371.5	589(589.55)	746.8	589(589.58)	997.2	617(618.64)	796.5
G25	589	589(589.67)	832.5	589(589.82)	895.9	589(589.81)	663.0	617(618.78)	764.1
G26	587	587(588.16)	313.9	587(588.27)	199.3	588(588.16)	1272.7	617(618.75)	350.5
G27	820	820(820.67)	384.7	820(820.94)	622.4	820(821.09)	513.43	873(875.45)	960.3
G28	820	**822**(822.98)	310.7	823(823.73)	533.1	823(823.89)	412.9	874(875.73)	1718.2
G29	820	820(820.89)	591.4	820(821.1)	195.7	821(821.55)	878.2	873(875.38)	1195.9
G30	821	**821**(821.95)	70.8	822(822.32)	673.8	822(822.42)	1085.3	873(875.65)	480.3
G31	819	**819**(819.98)	705.2	820(820.75)	474.3	820(820.88)	1082.4	874(875.39)	857.6
G32	40	40(40)	0.3	40(40)	0.3	40(40)	0.3	40(40)	0.5
G33	50	50(50)	0.2	50(50)	0.1	50(50)	0.1	50(50)	4.9
G34	80	80(80)	20.3	80(81.8)	0.1	80(82.2)	0.0	80(81.07)	1541.0
G35	436	**436**(438.49)	408.9	439(440.24)	1571.9	439(440.81)	2192.9	606(615.47)	1862.3
G36	441	**441**(443.01)	2112.7	442(444.19)	734.3	442(445.21)	961.7	612(620.99)	463.6
G37	435	**435**(437.05)	858.4	436(438.99)	1065.2	437(439.26)	1457.2	608(616.04)	844.0
G38	439	**439**(441.22)	1232.0	441(443.38)	321.4	440(443.65)	65.5	608(618.98)	2438.7
G39	436	**436**(438.34)	90.8	438(440.26)	285.3	439(441)	920.2	606(615.46)	1838.8
G40	440	**440**(442.86)	2151.2	442(444.49)	972.7	442(445.01)	969.0	615(621.22)	1396.0
G41	435	**435**(436.94)	1343.9	437(439.2)	1172.9	437(439.4)	342.5	607(616.24)	835.1
G42	439	**439**(441.41)	991.7	441(443.51)	317.8	442(444.16)	1296.9	611(619.14)	2424.7
G43	411	411(411)	80.9	411(411.08)	875.3	411(411.16)	1151.4	434(435.22)	1171.5
G44	411	411(411.01)	759.3	411(411.12)	1072.6	411(411.32)	921.9	433(435.19)	1615.2
G45	410	410(410)	327.0	410(410)	342.1	410(410.01)	621.8	434(435.35)	191.2
G46	411	412(412)	7.9	412(412)	135.6	**411**(411.99)	77.3	434(435.68)	903.1
G47	411	411(411.98)	0.8	411(411.67)	664.5	411(411.9)	224.7	434(435.53)	433.0
G48	100	100(101.2)	53.0	100(101)	0.1	100(101.6)	0.1	101(104.93)	55.3
G49	60	60(60)	0.5	60(60)	0.4	60(60)	0.2	60(60)	88.9
G50	50	50(50)	1.3	50(50)	1.0	50(50)	0.5	50(50)	4.9
G51	224	224(224.66)	721.6	224(224.64)	672.4	224(224.75)	691.0	295(299.53)	1967.0
G52	223	223(224.32)	347.1	223(224)	914.7	223(223.99)	1144.9	294(300.05)	936.8
G53	221	221(222.09)	541.4	221(221.55)	901.8	221(221.6)	945.5	293(299.63)	1675.6
G54	219	219(219.31)	888.8	219(219)	437.2	219(219)	395.1	292(299.27)	2707.6
Total		53/54		42/54		43/54		7/54	

Acknowledgment

We are grateful to the referees for their comments and questions which helped us to improve the paper. This work was partially supported by the Region of "Pays de la Loire" (France) within the RADAPOP and LigeRO Projects.

References

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

[Balas and de Souza, 2005b] E. Balas and C. C. de Souza. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 103:609–631, 2005.

[Battiti and Protasi, 2001] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[Benlic and Hao, 2011] U. Benlic and J.K. Hao. A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*, 15(5):624–642, 2011.

[Benlic and Hao, 2012] U. Benlic and J.K. Hao. A study of breakout local search for the minimum sum coloring problem. In *SEAL-2012*, Lecture Notes in Computer Science 7673:128–137, 2012.

[Benlic and Hao, 2013a] U. Benlic and J.K. Hao. Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206, 2013.

[Benlic and Hao, 2013b] U. Benlic and J.K. Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013.

[Benlic and Hao, 2013c] U. Benlic and J.K. Hao. Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence*, 26(3):1162-1173, 2013.

[Biha and Meurs, 2011] M D. Biha and M.J. Meurs. An exact algorithm for solving the vertex separator problem. *Journal of Global Optimization*, 49(3):425–434, 2011.

[Bui and Jones, 1992] T.N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3):153–159, 1992.

[Cavalcante and de Souza, 2011] V. F. Cavalcante and C. C. de Souza. Exact algorithms for the vertex separator problem in graphs. *Networks*, 57:212–230, 2011.

[Fiduccia and Mattheyses, 1982] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181. IEEE Press, 1982.

[Fukuyama, 2006] J. Fukuyama. Np-completeness of the planar separator problems. *Journal of Graph Algorithms and Applications*, 4:317–328, 2006.

[Glover, 1989] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–260, 1989.

[Helmberg and Rendl, 2000] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2000.

[Lourenco *et al.*, 2003] H.R. Lourenco, O. Martin, and T. Stützle. *Iterated local search, Handbook of Metaheuristics*. Springer-Verlag, Berlin Heidelberg, 2003.

[Woodruff and Zemel, 1993] D. L. Woodruff and E. Zemel. Hashing vectors for tabu search. *Annals of Operations Research*, 41(2):123–137, 1993.