Path relinking for the vertex separator problem

Fuda Ma[a], Yang Wang[c], Jin-Kao Hao[a,b,*]

[a]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[b]*Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France*
[c]*School of Management, Northwestern Polytechnical University, 127 Youyi West Road, 710072 Xi'an, China*

Expert Systems with Application, DOI:10.1016/j.eswa.2017.03.064

Abstract

This paper presents the first population-based path relinking algorithm for solving the NP-hard vertex separator problem in graphs. The proposed algorithm employs a dedicated relinking procedure to generate intermediate solutions between an initiating solution and a guiding solution taken from a reference set of elite solutions (population) and uses a fast tabu search procedure to improve some selected intermediate solutions. Special care is taken to ensure the diversity of the reference set. Dedicated data structures based on bucket sorting are employed to ensure a high computational efficiency. The proposed algorithm is assessed on four sets of 365 benchmark instances with up to 20,000 vertices, and shows highly comparative results compared to the state-of-the-art methods in the literature. Specifically, we report improved best solutions (new upper bounds) for 67 instances which can serve as reference values for assessment of other algorithms for the problem.

Keywords: Vertex separator; Graph partitioning; Path relinking; Population-based heuristics.

1. Introduction

Given an undirected graph G (which may be disconnected) with a vertex set $V = \{v_1, \ldots, v_n\}$ where each vertex v_i is associated with an non-negative weight w_i and an unweighted edge set E, the vertex separator problem (VSP) is to partition V into three disjoint subsets A, B and C, where A and B are non-empty, such that the total weight of vertices in C is minimized subject to two constraints: (i) there is no edge between A and B and (ii) the cardinality of A and B does not exceed a given positive integer b. Set C is called the separator

*Corresponding author
Email addresses: `ma@info.univ-angers.fr` (Fuda Ma), `yangw@nwpu.edu.cn` (Yang Wang), `hao@info.univ-angers.fr` (Jin-Kao Hao)

of G while A and B are called the shores of the separator. Formally, VSP is formulated as follows.

$$
\begin{aligned}
\min \quad & \sum_{i \in C} w_i && (1)\\
\text{subject to} \quad & C = V \setminus (A \cup B), (A \times B) \cap E = \emptyset, A \cap B = \emptyset && (2)\\
& \max\{|A|, |B|\} \leq b && (3)\\
& A \neq \emptyset, B \neq \emptyset, A, B, C \subset V && (4)
\end{aligned}
$$

where constraint (2) ensures that no edge exists for any pair of vertices between shores A and B and constraint (3) requires both A and B contain no more than b vertices. A separator C is considered as balanced if $\max\{|A|, |B|\} \leq 2|V|/3$.

One of the main and first applications of the VSP concerns sparse matrix re-orderings (George & Liu, 1981). Other applications include, for instance, detection of brittle nodes in telecommunication networks (Biha & Meurs, 2011), identification of the minimal separator in the divide-and-conquer based graph algorithms (Evrendilek, 2008; Lipton & Tarjan, 1979) as well as finding protein conformation in bioinformatics (Fu & Chen, 2006). From the point view of computational complexity, the VSP is known to be NP-hard for general graphs (Bui & Jones, 1992) and even for planar graphs (Fukuyama, 2006).

As general solution methods, Leighton (1983) presented an approximation algorithm based on a linear relaxation technique and achieved an approximation ratio of $O(log\ n)$. Feige et al. (2008) improved this result to $O(\sqrt{log\ n})$ by utilizing a semidefinite relaxation method.

There are several exact algorithms, which are able to solve instances with up to a few hundred of vertices. In 2005, de Souza & Balas (2005) designed a branch-and-cut algorithm which explores valid polyhedral inequalities obtained in Balas & de Souza (2005) and conducted extensive computational experiments. In 2011, de Souza & Cavalcante (2011) proposed a hybrid algorithm that combines Lagrangian relaxation with cutting plane techniques. Computational results showed that the hybrid algorithm outperforms the best exact algorithm available. In 2011, Biha & Meurs (2011) presented an exact approach based on a new class of valid inequalities and performed comparisons with the algorithm in de Souza & Balas (2005).

In addition to the above approximation and exact approaches, heuristic and metaheuristic algorithms have been devised to obtain good quality solutions for large VSP instances in reasonable computing times. We summarize the state-of-the-art heuristic algorithms in the literature as follows.

In 2013, Benlic & Hao (2013) presented the breakout local search (BLS) algorithm which combines a local search procedure with an adaptive perturbation procedure. The local search procedure uses a dedicated move operator to transform the current solution to a neighbor solution. This is achieved by displacing a vertex v from the separator C to the shore subset A or B, followed by displacing all the adjacent vertices of v from the opposite shore subset to the separator

C. The perturbation procedure employs an adaptive mechanism to apply either a directed perturbation or a random perturbation to escape local optimum traps and direct the search toward unexplored areas. Experimental results on benchmark instances with up to 3000 vertices demonstrated the effectiveness of the BLS method.

In 2014, Sánchez-Oro et al. (2014) introduced several variable neighborhood search (VNS) algorithms, which alternate between a local search phase and a shaking phase. Two initial solution constructive procedures (random and greedy) are used to generate seeding solutions. The local search phase relies on three types of basic moves and two combined moves to attain a local optimum. A variable neighborhood descent procedure is then used to further improve the encountered solution with the two combined neighborhoods. The shaking phase carries out random perturbations to produce new starting feasible solutions. Experiments on benchmark instances with up to 1000 vertices showed the effectiveness of the VNS algorithms.

In 2015, Hager & Hungerford (2015) proposed a continuous optimization approach. The VSP problem is first formulated as a continuous bilinear quadratic program, which is then solved by a multilevel algorithm. Following the general multilevel graph approach, the proposed algorithm is composed of three phases including 1) a coarsening phase that hierarchically coarsens a graph into a sequence of smaller graphs; 2) a refinement phase that finds an initial solution to the graph in the coarsest level; and 3) an uncoarsening phase that projects the solution of the lower-level graph to its upper level graph. Both hill climbing and Fiduccia–Mattheyses heuristics are used to solve each hierarchy of graphs. Experiments showed that this approach outperforms the general graph partitioning package METIS in terms of solution quality for graphs with 1000 to 5000 vertices, but is outperformed by the BLS method (Benlic & Hao, 2013).

Recently, the population-based path relinking framework (Glover & Laguna, 1997; Glover, 1998) has attracted much attention in combinatorial optimization and intelligent problem solving. The approach has shown outstanding performances in solving a number of challenging decision and optimization problems in various settings, such as unconstrained binary quadratic optimization (Wang et al., 2012), flow shop sequencing and scheduling (Costa et al., 2012; Peng et al., 2015; Zeng et al., 2013), clustering (Martins de Oliveira et al., 2014), web services composition (Parejo et al., 2014), frequency assignment (Lai & Hao, 2015) and quadratic multiple knapsack (Chen et al., 2016). PR has also been combined with other metaheuristics such as genetic algorithms (Vallada & Ruiz, 2010), scatter search (González et al., 2015) and GRASP (Mestria et al., 2013) to solve several difficult combinatorial problems.

In this work, we are interested in advancing the state-of-the-art of solving the VSP with heuristics. For this purpose, we propose the first population-based path relinking algorithm for the VSP (named PR-VSP). We identify the main contributions of this work as follows.

- The proposed PR-VSP algorithm is the first adaptation of the general evolutionary path-relinking framework to the NP-hard vertex separator problem. To ensure its search efficiency, PR-VSP combines a fast solution improvement procedure with a dedicated path relinking method. The solution improvement procedure relies on two complementary neighborhood search operators to visit promising candidate solutions while the relinking method employs a distanced-based strategy to generate new solutions. Additionally, special care is taken to ensure the diversity of the reference set (or population) of elite solutions. Dedicated data structures based on bucket sorting are employed to ensure a high computational efficiency.

- The performance of the proposed algorithm is assessed on four sets of 365 benchmark instances (with up to 20,000 vertices) commonly used in the literature and compared with state-of-the-art VSP algorithms. The computational results show that PR-VSP competes very favorably compared to the current best performing algorithms in terms of solution quality and computing efficiency. Specifically, the proposed algorithm finds new best solutions (updated upper bounds) for 67 instances and matches previous best solutions for all but one instance. The new upper bounds are particularly useful for assessment of other VSP algorithms.

The reminder of the paper is organized as follows. Section 2 presents the general scheme and each component of the proposed PR-VSP. Section 3 is dedicated to experimental results and comparisons with state-of-the-art algorithms in the literature. Concluding remarks are given in Section 4.

2. The proposed path relinking algorithm for VSP

Path relinking is a population-based general framework which was originally proposed for enhancing the tabu search method (Glover & Laguna, 1997; Glover, 1998). Like other general metaheuristics, when applying such a method to a particular problem, it is necessary and indispensable to make a number of specific adaptations to the problem under consideration (Wang et al., 2017; Wang & Punnen, 2017). In this section, we first expose the main scheme of the proposed algorithm and then explain each specific component.

2.1. Main scheme

Algorithm 1 shows the general scheme of the PR-VSP algorithm. It first creates a reference set $RefSet$ consisting of a set of elite (feasible) solutions $\{S_1, S_2, \ldots, S_p\}$ and constructs a set $PairSet$ composed of indexes of all pairwise solutions in $RefSet$ (See Alg. 2, Section 2.3). Then, for each pair of solutions (S_i and S_j), a relinking method is utilized to build a solution path (i.e., a sequence of intermediate solutions) that connects the initiating solution where the path starts from (say S_i) and the guiding solution where the path ends (say S_j) (see Section 2.5). By interchanging the initiating and guiding solutions, another path is built in the same way. A solution selection method (see Section 2.6) is then applied to pick one or multiple solutions from the path for further improvement by the iterated tabu search method (see Section 2.4). The improved solution

Algorithm 1 Outline of the path relinking algorithm

```
1: Input: an undirected graph $G = (V, E)$ with its vertex weight vector, an upper limit
   $b$ for the size of each shore subset, an integer $p$ for the $RefSet$ size
2: Output: the best solution $S^*$ found and its objective value $f(S^*)$
3: repeat
4:     Initialize $RefSet$ and $PairSet$ with $|RefSet| = p$ (see Section 2.3)
5:     Record the best solution $S^*$ in $RefSet$ and the objective value $f(S^*)$
6:     while ($PairSet \neq \emptyset$) do
7:         Pick an index pair $(i, j) \in PairSet$ to get a pair of solutions $(S^i, S^j)$ from
           $RefSet$
8:         Apply the Relinking Method to build a path from $S^i$ to $S^j$ and another path
           from $S^j$ to $S^i$ (see Section 2.5)
9:         Apply the Solution Selection Method to select solutions on each path (see Section
           2.6)
10:        Apply the Solution Improvement Method to the selected solutions (see Section
           2.4)
11:        Update the best solution $S^*$ and its objective value $f(S)^*$
12:        Update $RefSet$ and $PairSet$ (see Section 2.3)
13:    end while
14: until A stopping condition (e.g., a cutoff time limit) is met
```

is then used to update $RefSet$, $PairSet$ and the best solution found S^* (see Section 2.3). When $Pairset$ becomes empty, the algorithm re-initializes $RefSet$ and then repeats the whole procedure until a stopping condition (e.g., a cutoff time limit) is reached.

2.2. Search space

Given $G = (V, E)$, a candidate solution to the VPS is any partition of the vertex set V into a separator C and two shores A and B satisfying constraints (2), (3) and (4) given in the introduction. Thus, we define the search space Ω explored by the PR-VSP algorithm to be the set of all such possible three-way partitions $\{A, B, C\}$ of V, i.e.,

$$\Omega = \{\{A, B, C\} : A, B \subset V, C = V \setminus (A \cup B), (A \times B) \cap E = \emptyset, \\ A \cap B = \emptyset, \max\{|A|, |B|\} \leq b\}, A \neq \emptyset, B \neq \emptyset. \quad (5)$$

For a given candidate solution $S = \{A, B, C\}$ of Ω, its quality is directly given by its objective value, i.e., the weight sum of the vertices in the separator C, $f(S) = \sum_{i \in C} w_i$. For two candidate solutions S' and S'' in the search space, S' is better than S'' if and only if $f(S') < f(S'')$.

Notice that for a graph of reasonable size (say several hundreds of vertices), the number of possible solutions in Ω can be already quite large. Moreover, the search space Ω will increase very rapidly with the increase of the number of vertices of the graph. The purpose of the proposed PR-VSP algorithm is to locate a solution as good as possible in this highly combinatorial search space within a given computing effort. To reach this goal, PR-VSP calls for a number of dedicated search operators and strategies that are explained below.

2.3. RefSet and PairSet initialization and updating

Algorithm 2 $RefSet$ and $PairSet$ Initialization

```
1: Input: an undirected graph $G = (V, E)$ and an integer $p$ for the size of $RefSet$
2: Output: reference set $RefSet$ composed of $p$ elite solutions, an index pair set $PairSet$
   consisting of index pairs of all pairwise solutions in $RefSet$
3: $iter \leftarrow 0$
4: repeat
5:    $\{A, B\} \leftarrow RandAssign(V)$ /* Randomly assign all vertices $v \in V$ into the shore
      subsets A and B*/
6:    for each $(v_i, v_j) \in E$ where $v_i \in A, v_j \in B$ do
7:       $v^* \leftarrow RandSelect(\{v_i, v_j\})$ /* Randomly select a vertex $v^*$ from $\{v_i, v_j\}$ */
8:       Displace $v^*$ to the separator $C$
9:    end for
10:   for each $X \in \{A, B\}$ do
11:      while $|X| > b$ do
12:         $v^* \leftarrow RandSelect(X)$ /* Randomly select a vertex $v^*$ from $X$ */
13:         Displace $v^*$ to $C$
14:      end while
15:   end for
16:   $S \leftarrow \{A, B, C\}$
17:   $S_{iter} \leftarrow Tabu\_search(S)$ /* $S_{iter}$ is the best solution found by tabu search and is
      considered as a candidate initial solution */
18:   Set $iter = iter + 1$
19: until $iter \geq 2p$
20: $RefSet \leftarrow SelectBestSolutions(p, \{S_0, S_1, ..., S_{2p-1}\})$ /* Select $p$ non-identical so-
    lutions with the best objective values */
21: $PairSet \leftarrow \{\{i, j\} : S_i \in RefSet, S_j \in RefSet, i < j\}$
```

The reference set $RefSet$ contains the working solutions of the PR-VSP algorithm and is composed of p elite solutions (See Alg. 1, line 4). $RefSet$ is created by employing a randomized initialization procedure to acquire diverse solutions and a tabu search based solution improvement method to assure high quality of the acquired solutions (See Alg. 2). Each initial solution is generated by the procedure presented in Benlic & Hao (2013), which applies the following steps. First, we randomly assign the vertices into the shore subsets A and B (See Alg. 2, line 5). Then, for each cutting edge $(v_i, v_j) \in E$ such that $v_i \in A$ and $v_j \in B$, we displace randomly v_i or v_j to the separator C (See Alg. 2, lines 6 - 9). Finally, if a shore has a cardinality that surpasses the upper limit b, we randomly displace vertices from the shore into the separator C until the upper limit constraint is satisfied (See Alg. 2, lines 10 - 15). Once a new solution is generated, it is immediately improved by the tabu search procedure of Section 2.4 (See Alg. 2, line 17). We repeat the above procedure to produce $2p$ improved solutions, from which p non-identical solutions with the best objective values are chosen to form $RefSet$ (See Alg. 2, line 19).

The $RefSet$ updating procedure decides the way of inserting a newly generated solution in $RefSet$ and removing an existing solution from $RefSet$ (See Alg. 1, line 12). To maintain a healthy $RefSet$, the updating mechanism (See Alg. 3) requires that the new solution S_n considered for insertion satisfies both a specified distance threshold τ and a solution quality criterion (Lai & Hao, 2015). Specifically, we first determine a solution S_c in $RefSet$ such that S_c has the minimum distance d_{min} to the solution S_n, the distance between S_c and S_n being the number of vertices not shared in the two separators (See Alg. 3, line 3). If $d_{min} \le \tau$, then S_n replaces the solution S_c if S_c is no better than S_n; otherwise S_n is directly discarded. If $d_{min} > \tau$, then S_n replaces the worst solution S_w in $RefSet$ if S_n is no worse than S_w or is discarded otherwise (See Alg. 3, lines 4 - 12). The complexity of each $RefSet$ updating operation is $O(p \cdot |C|)$.

Algorithm 3 $RefSet$ Update

```
1: Input: $G = (V, E)$, $RefSet$, a new solution $S_n$ considered for insertion, the worst
   solution $S_w$ in $RefSet$
2: Output: $RefSet$
3: $S_c, d_{min} \leftarrow FindMinDistanceSolution(RefSet, S_n)$
4: if $d_{min} \le \tau$ then
5:     if $f(S_n) \le f(S_c)$ then
6:         Add $S_n$ to $Refset$ and remove $S_c$ from $RefSet$
7:     end if
8: else
9:     if $f(S_n) \le f(S_w)$ then
10:        Add $S_n$ to $Refset$ and remove $S_w$ from $RefSet$
11:    end if
12: end if
```

$PairSet$ is used to mark each pairwise solutions which will experience a path relinking procedure (See Alg. 1, lines 4 and 7). It is initialized as the index pair of each pair of solutions in $RefSet$ (See Alg. 2, line 20). Each time an index pair experiences a path relinking, it is removed from $PairSet$ (See Alg. 4, line 3). Moreover, as shown in Algorithm 4, if a newly produced solution replaces a solution in $RefSet$, all the index pairs related to this replaced solution are removed from $PairSet$ and new index pairs composed of the new solution and each other solution in $RefSet$ are added into $PairSet$. When $RefSet$ is not updated for a certain consecutive number of times, all the index pairs are removed and $PairSet$ becomes empty.

2.4. The solution improvement method

Within the proposed PR-VSP algorithm, we use an iterated tabu search (ITS) procedure as the solution improvement method. Basically, this ITS procedure alternates between a tabu search phase (Glover & Laguna, 1997) and a perturbation phase. Each tabu search phase stops when the best solution is not improved for a consecutive number of iterations (called *iteration cutoff*, set

Algorithm 4 *PairSet* Update

```
1: Input: $PairSet$, the removed solution $S_k$ from $RefSet$
2: Output: an updated $PairSet$
3: Remove the used index pair in the current path relinking procedure from $PairSet$
4: for each $S_i \in RefSet$ do
5:     if $i < k$ & $(i,k) \notin PairSet$ then
6:         $PairSet = PairSet \cup \{(i,k)\}$
7:     end if
8:     if $i > k$ & $(i,k) \notin PairSet$ then
9:         $PairSet = PairSet \cup \{(k,i)\}$
10:    end if
11: end for
```

as $\beta * |C|$ where β is a parameter). At this moment, the perturbation phase is triggered to generate a perturbed solution which serves as the starting solution of the next ITS run. The following sections describe the key components of the ITS procedure.

2.4.1. Moves and calculation of move gain

As explained in Section 2.2, a candiate solution of the VSP is a partition $S = \{A, B, C\}$ of the vertex set V satisfying the problem constraints ($A \neq \emptyset, B \neq \emptyset$, $(A \times B) \cap E = \emptyset$ and $\max\{|A|, |B|\} \leq b$). To generate neighbor solutions from the current solution, the following two move operators are employed.

The first move operator (called *1-move*) displaces a vertex v_i from the separator C to a shore subset A or B, without violating the constraint $\max\{|A|, |B|\} \leq b$. To ensure the constraint $(A \times B) \cap E = \emptyset$, a repair operation is followed to displace to the separator C all the vertices in the opposite shore which are adjacent to v_i. This *1-move* operator has been used in various algorithms (Ashcraft & Liu, 1994; Benlic & Hao, 2013; Sánchez-Oro et al., 2014). The objective gain of performing a *1-move* (i.e., the objective variation between its neighbor solution and the current solution S, also called move gain) is calculated as

$$mg^1(v_i, S) = \begin{cases} -w_i + \sum_{v_j \in B, (v_i, v_j) \in E} w_j & \text{if } v_i \in C \text{ moves to } A \\ -w_i + \sum_{v_j \in A, (v_i, v_j) \in E} w_j & \text{if } v_i \in C \text{ moves to } B \end{cases} \tag{6}$$

The second move operator (called *swap-move*) is a new operator introduced in this work, which is designed to handle the case where the size of a shore subset reaches the upper limit b (i.e., $|A| = b$ or $|B| = b$). The *swap-move* operator displaces a vertex v_i from the separator C to the shore subset whose size is equal to the upper limit b (thus momentarily violating the constraint $\max\{|A|, |B|\} \leq b$) and then displaces another vertex w_{min} with the minimum weight from this shore subset to the separator C (to re-establish the constraint $\max\{|A|, |B|\} \leq b$). To satisfy the constraint $(A \times B) \cap E = \emptyset$, the same repair operation as for *1-move* is employed. The objective gain of performing a *swap-move* operation is calculated according to Eq. (7).

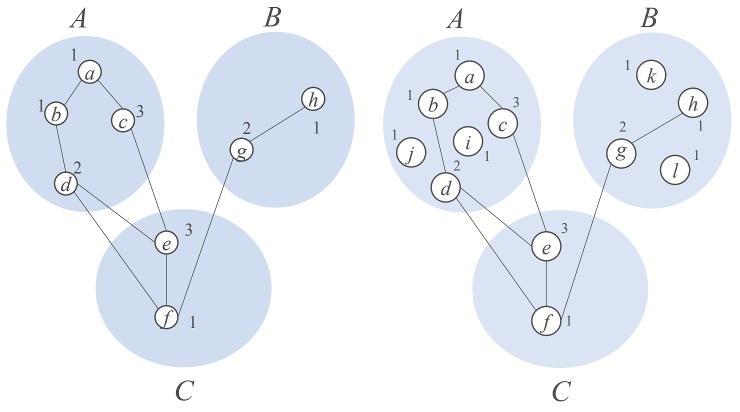

Figure 1: Two examples showing the benefit of the *swap-move* operator

$$mg^2(v_i, S) = \begin{cases} -w_i + w^A_{min} + \sum_{v_j \in B, (v_i, v_j) \in E} w_j & \text{if } |A| = b \text{ and } v_i \in C \text{ moves to } A \\ -w_i + w^B_{min} + \sum_{v_j \in A, (v_i, v_j) \in E} w_j & \text{if } |B| = b \text{ and } v_i \in C \text{ moves to } B \end{cases} \tag{7}$$

To show the interest of the newly introduced *swap-move* operator with respect to the conventional *1-move* operator, let us consider two illustrative examples (Fig. 1).

The left graph in Fig. 1 ($|V| = 8$ and $b = 4$) shows a candidate solution $S = \{A = \{a, b, c, d\}, B = \{g, h\}, C = \{e, f\}\}$ with an objective value of 4 ($f(S) = 4$). If we use *1-move* to displace vertex $e \in C$, then e must be moved from C to B since the number of vertices in A has already reached the given upper limit b. Once e is displaced in B, the repair operation displaces its adjacent vertices c and d from A to C. Therefore, the move gain obtained by this *1-move* operation is $-w_e + w_c + w_d = -3 + 2 + 3 = 2$ (i.e., the objective function value of the resulting solution S' is $f(S') = f(S) + 2 = 6$). However, if we apply *swap-move* to exchange vertex e from separator C against a from shore A, the resulting solution S'' has an objective gain of $-w_e + w_a = -2$, leading to a better objective value of $f(S'') = f(S) - 2 = 2$.

The right graph in Fig. 1 ($|V| = 12$ and $b = 6$) shows a solution $S = \{A = \{a, b, c, d, i, j\}, B = \{g, h, l, k\}, C = \{e, f\}\}$ which includes 4 isolated vertices $I = \{i, j, k, l\}$. If we use *1-move* to displace e from C to B, the resulting solution S' gets an objective *increase* of 2. Note that *1-move* can in no way move any vertex of I into the separator C since the vertices of I are not connected to any other vertex (including those of C). On the other hand, we can use *swap-move* to exchange e against the vertex i or j to obtain an improved solution with an objective *decrease* of 2.

It is noted that using *swap-move* is particularly useful when the graph con-

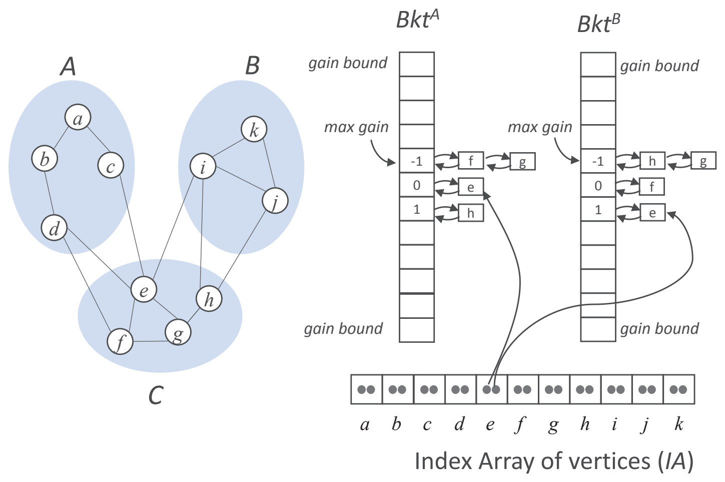

Figure 2: An example of the bucket structure for the vertex separator problem

tains isolated vertices or vertices with low degrees.

2.4.2. Bucket sorting

To quickly calculate the move gain for a *1-move* and *swap-move*, we use an n-dimensional vector Δ^A, where each entry Δ_i^A records the total weight of all vertices of the shore subset A which are adjacent to a vertex v_i (i.e., $\Delta_i^A = \sum_{v_j \in A, (v_i, v_j) \in E} w_j$). With Δ_i^A preliminarily computed, the objective gains of performing both *1-move* and *swap-move* shown in Eq. (6) and (7) can be obtained in $O(1)$ time. Similarly, another vector Δ^B is employed in the same way for the shore subset B. By simply replacing all the occurrences of A by B, we obtain the updating equations of Δ^B.

In addition, a bucket sorting technique inspired from (Fiduccia & Mattheyses, 1982) is utilized to quickly identify the best move in $O(1)$ time instead of scanning all the move gains of vertices in the separator C. Specifically, we use two arrays of buckets Bkt^A and Bkt^B to record the objective gain of displacing any vertex from the separator C to each shore subset A or B. Notice that when the condition for performing a *swap-move* is satisfied, the corresponding entry in the bucket actually represents the objective gain of *swap-move*. In each bucket array, the qth entry stores all the vertices with the objective gain currently equaling to q, which are managed by a doubly linked list. To ensure a direct access to the vertex in the doubly linked list, another index array is also employed, in which each entry stores the address that points to its vertex in the doubly linked list. For each array of buckets, identifying the best vertex with the maximum objective gain equals to the identification of the first non-empty bucket from the top of the bucket array, from which a vertex is randomly chosen from the doubly linked list.

Fig. 2 shows an illustrative example of the bucket structure for the VSP.

The graph (Fig. 2, left) has 11 vertices, where all the vertices have a weight of 1 for simplicity. Given the solution $S = \{A = \{a, b, c, d\}, B = \{i, j, k\}, C = \{e, f, g, h\}\}$, the bucket sorting structure is shown in Fig. 2 (right). To see how the vertices are arranged in the structure, we consider vertex e as an example. The move gain of displacing e from C to A is calculated as $-w_e + \Delta_e^B = 0$, so e is stored in the position $p = 0$ of the bucket array Bkt^A. In the same token, the vertex index e is stored in the position $p = -1$ of the bucket array Bkt^B. For each vertex in the separator C, we store it in the right entries of the buckets Bkt^A and Bkt^B in the same way. In addition, each entry of the index array of vertices shown at the bottom of Fig. 2 points to the position of this vertex in the buckets Bkt^A and Bkt^B. From the top of buckets, we see that displacing f to A, h to B, g to A and g to B leads to the same maximum gain of -1, from which one move will be chosen at random.

To perform a *1-move* or *swap-move* operation, the following three steps are concerned: 1) a vertex is displaced from the separator to either shore subset; 2) the adjacent vertices in the opposite shore subset of the displaced vertex are displaced to the separator; 3) a vertex is displaced from a shore subset to the separator. Now, let us take the shore subset A as an example to illustrate how to quickly update the Δ^A vector.

- If a vertex v_i is displaced from C to A, the Δ^A vector is updated as $\Delta_j^A = \Delta_j^A + w_i$, for all $v_j \in V$ where $(v_i, v_j) \in E$
- Let $N^A(i)$ denote the set of the vertices in A that are adjacent to the vertex $v_i \in B$. If all the vertices in $N^A(i)$ are displaced from A to C, Δ^A is updated as $\Delta_j^A = \Delta_j^A - \sum_{v_k \in N^A(i), (v_k, v_j) \in E} w_k$, for all $v_j \in V$
- If a vertex v_i is displaced from A to C, Δ^A is updated as $\Delta_j^A = \Delta_j^A - w_i$, for all $v_j \in V$ where $(v_i, v_j) \in E$

The method to update Δ^B for the operations on the shore subset B is obtained by replacing all the appearances of A by B and B by A.

The operations on the bucket structure with regard to the above mentioned move operations are as follows.

- Delete: delete a vertex from Bkt^A and Bkt^B if it is displaced from C to A
- Add: add a vertex into Bkt^A and Bkt^B if it is displaced from A to C
- Shift: shift a vertex to the correct entry in each bucket array according to its updated objective gains

Our experimental results indicate that the devised bucket sorting technique considerably improves the computational efficiency, thus the performance of our path relinking algorithm.

2.4.3. Tabu search

The tabu search phase uses both the *1-move* and *swap-move* operators to exploit the search space. At each iteration, TS performs a best move among the set of eligible moves. A move is eligible if it is not forbidden by the tabu list (see below), or if it leads to a solution better than any solution visited so far. Precisely, if the size of each shore subset is less than the upper limit b, then only the *1-move* operator is used during the search. Otherwise, both *1-move* and *swap-move* have a chance to be applied. Specifically, if the objective gain of performing a *swap-move* is better than that of performing a *1-move*, then each type of move will be selected with an equal probability of 50%. This rule is overridden if performing a *swap-move* leads to a solution better than the best solution found so far. In this case, the *swap-move* is always performed. The idea to take a worse *1-move* into consideration is to reduce the shore subset whose size reaches b, which to some extent enhances the search diversification. Note that if a shore subset becomes empty during a certain tabu search iterations, the next iteration will force a vertex to be displaced to this empty subset. In this way, we assure that the tabu search focuses its exploration on the feasible search area.

Tabu search uses a short memory (called tabu list) to prohibit recently displaced vertices from being moved again for the next tt iterations (called tabu tenure) (Glover & Laguna, 1997). The tabu tenure is adaptively tuned according to the search status. Specifically, let C be the separator and $dmax$ denote the average value of the highest 5% vertex degrees, then the tabu tenure is set as $tt = min\{dmax, |C|/2\} + min\{Rand(\alpha \times dmax), |C|/2\}$, where $Rand(\alpha \times dmax)$ returns a random integer no greater than $\alpha \times dmax$ and α is a parameter. For a *1-move* which displaces a vertex v_i from C to A, given that v_i may go back to C due to the change of vertices in B, we prohibit v_i from joining A for the next tt iterations. For a *swap-move* which exchanges a vertex v_i of C against a vertex v_j of A, we prohibit both v_i and v_j from moving to A for the next tt iterations. The other vertices involved in a move are not concerned by the tabu list.

2.4.4. Perturbation

The perturbation phase performs a number of consecutive *1-move* operations on the local optimum from the last tabu search phase. Specifically, each perturbation step randomly displaces a vertex from the separator C to either shore subset with equal probability, followed by a repair operation to make the resulting solution feasible if needed. A large number of perturbation moves changes a large part of the input solution while a small number of perturbation moves may fail to lead the search to escape the attractor around the local optimum. We set experimentally the number of perturbation moves as $\rho * |C|$, where $\rho < 0.5$ is a parameter.

2.5. The relinking method

The relinking method constructs a solution path connecting an initiating solution and a guiding solution (both from $RefSet$), where each intermediate solution on the path gradually incorporates attributes from the guiding solution and finally matches the guiding solution (Glover, 1998). Algorithm 5 shows the outline of our proposed path relinking method for the vertex separator problem.

It first calculates the distance between the initiating solution and the guiding solution (See Alg. 5, lines 3 - 4). For two solutions $S_i = (A_i, B_i, C_i)$ and $S_g = (A_g, B_g, C_g)$, let SD denote the symmetric difference of the sets C_i and C_g, i.e., $SD = C_i \Delta C_g = (C_i \cup C_g) \setminus (C_i \cap C_g)$. The distance d between S_i and S_g is defined as the cardinality of SD, i.e., by $d = |SD|$.

Then a sequence of intermediate solutions $S(1), S(2), \ldots, S(d-1)$ from the initiating solution $S_i = \{A_i, B_i, C_i\}$ to the guiding solution $S_g = \{A_g, B_g, C_g\}$ are produced by using operations OP_1 and OP_2 (See Alg. 5, lines 5 - 19). For vertices in $C_i \setminus C_g$, OP_1 moves a vertex from C_i to A_i, if it is in A_g, or moved to B_i, if it is in B_g. For vertices in $C_g \setminus C_i$, OP_2 displaces a vertex from its current shore subset in the solution S_i to the separator C_i. repair operation is needed when an infeasible solution $S(t)$ on the path relinking step is obtained by performing the OP_1 operation. The method of repairing $S(t)$ to a feasible solution $\hat{S}(t)$ is to move all the adjacent vertices in the opposite shore subset of the displaced vertex v_i to the separator C_i (See Alg. 5, line 10). This can be realized in $O(|V| \cdot c)$ where $c = \frac{2|E|}{|V|(|V|-1)}$ is the density of the graph $G(V, E)$. Suppose that two consecutive OP_1 operations (moving the vertex v_k from C_i to A_i and moving the vertex v_l from C_i to B_i) are performed on the path solution $S(t)$ to obtain the path solutions $S(t+1)$ and $S(t+2)$. To get the feasible solution $\hat{S}(t+1)$, the solution $S(t+1)$ is repaired by displacing all the adjacent vertices of v_k from B_i to C_i. To get the feasible solution $\hat{S}(t+2)$, the solution $S(t+2)$ is repaired by displacing first all the adjacent vertices of v_k from B_i to C_i and then all the adjacent vertices of v_l from A_i to C_i. Note that if the repaired solution $\hat{S}(t+2)$ is obtained from $\hat{S}(t+1)$, then only the latter repair operation that displaces all the adjacent vertices of v_l from A_i is needed.

Each step t for building the path selects a vertex from SD such that it results in a feasible solution with the best objective gain after performing OP_1 and OP_2 operations. This can be achieved in $O(|SD| \cdot |V| \cdot c)$. Each time a vertex in SD is displaced, it is deleted from SD and the distance between the resulting path solution and the guiding solution is decreased by 1. The next solution $S(t+1)$ is obtained by performing an OP_1 or OP_2 operation on the solution $S(t)$. After $d-1$ steps, the path relinking method terminates and the sequence of intermediate solutions on the path are obtained.

Fig. 3 provides an example to illustrate the relinking procedure. Two solutions $S_i = \{A_i = \{a, e, g, i\}, B_i = \{b, f\}, C_i = \{c, d, h\}\}$ and $S_g = \{A_g = \{a, e, g, h\}, B_g = \{d, f\}, C_g = \{b, c, i\}\}$ are given. To build a path starting from the solution S_i (initiating solution) and ending at the solution S_g (guiding solution), we first identify the symmetric difference $SD = \{b, d, h, i\}$ between the separators C_i and C_g. Then for each step in the relinking procedure, a vertex from SD goes through a OP_1 or OP_2 operation. Hence, four vertices can be chosen in the first relinking step, by displacing the vertex b from B_i to C_i, d from C_i to B_i, h from C_i to A_i or i from A_i to

Algorithm 5 Relinking method

```
1: Input: an initiating solution $S_i$ and a guiding solution $S_g$
2: Output: path solutions $S(1), S(2),\ldots, S(d-1)$
3: Identify the symmetric difference set $SD$ between the separator subsets $C_i$ and $C_g$
4: Set $d = |SD|$, $S(0) = S_i$, $t = 1$
5: while $t < d$ do
6:     Set $S(t) = S(t-1)$, $g_{max} = -\infty$
7:     for each $v_m \in SD$ do
8:         if $v_m \in C_i \setminus C_g$ then
9:             Perform the operation $OP_1$ for the solution $S(t-1)$ to obtain the solution $S(t)$
10:            Repair the solution $S(t)$ to obtain a feasible solution $\hat{S}(t)$ and record the objective gain $g_m$
11:        else
12:            Perform the operation $OP_2$ for the solution $S(t-1)$ to obtain the solution $S(t)$ and record the objective gain $g_m$
13:        end if
14:        if $g_m > g_{max}$ then
15:            $v^* = v_m$, $S^* = S(t)$, $g_{max} = g_m$
16:        end if
17:    end for
18:    Set $S(t) = S^*$, $SD = SD \setminus \{v^*\}$, $t = t + 1$
19: end while
```

C_i, which produce four candidate path solutions. Among them the solution $S(1) = \{A_i = \{a, e, g, i\}, B_i = \{b, d, f\}, C_i = \{c, h\}\}$ is chosen as the path solution because it leads to a feasible solution with the best objective value. Then starting from $S(1)$, three vertices can be chosen in the second relinking step to create three candidate path solutions, from which the solution $S(2) = \{A_i = \{a, e, g\}, B_i = \{b, d, f\}, C_i = \{c, h, i\}\}$ is chosen to be on the path. After a total of three steps, the path relinking procedure stops.

2.6. The solution selection method

The solution selection method aims to identify solutions from the sequence of intermediate solutions produced by the relinking method for further improvement by iterative tabu search. In general, several solutions can be selected for improvement. Considering that the solutions on the path are quite close to each other, running ITS on multiple solutions would lead to the same locally optimal solution. Therefore, we just select one solution from the path with the best objective value.

3. Experimental results

This section is dedicated to a large experimental assessment of the proposed PR-VSP algorithm. For this purpose, we present computational results on four sets of benchmark instances and compare our results with those reported by the state-of-the-art algorithms in the literature.

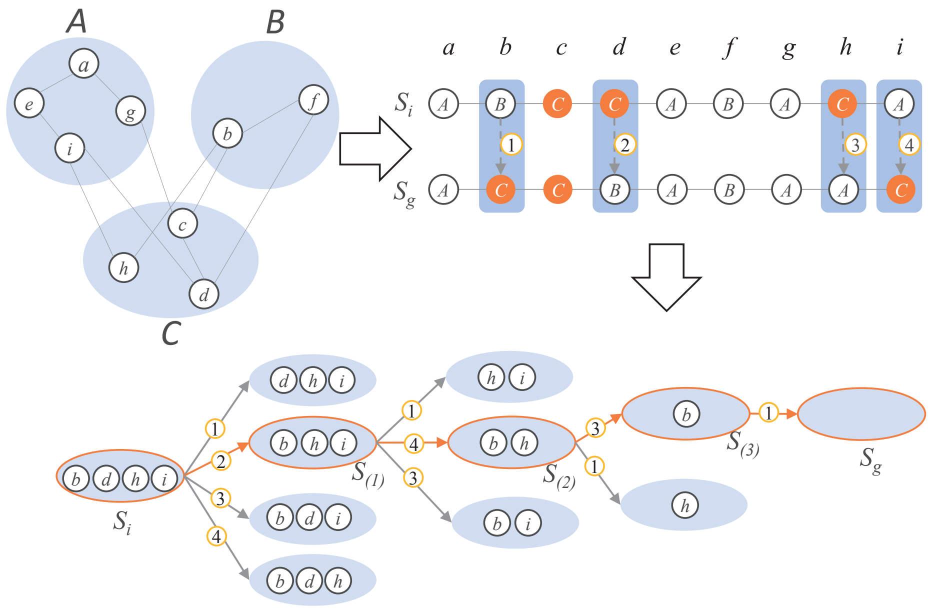

Figure 3: An illustrative example of the path relinking procedure

3.1. Experimental protocols

We use the following four sets of 365 benchmark instances which are commonly tested in the literature[1].

- **Traditional benchmarks:** This set[2] contains 104 small instances with $11 \leq |V| \leq 191$ and $20 \leq |E| \leq 13,992$ with known optimal solutions. This set of instances was first introduced and studied in de Souza & Balas (2005) and also tested in Benlic & Hao (2013); Biha & Meurs (2011).

- **Hermberg and Rendl benchmarks:** This set[3] is composed of 71 structured and random instances with $|V|$ ranging from 800 to 20,000 and graph density ranging from 0.000131 to 0.06. Note that the last 17 large graphs are investigated for the first time in this work. This set of instances was first tested in Benlic & Hao (2013).

- **Barabasi-Albert benchmarks:** This set[4] includes 95 instances with $100 \leq |V| \leq 1000$ and a random vertex degree in $[1, |V|]$. Graphs of this

[1]The solution certificates will be made available at http://www.info.univ-angers.fr/ hao/prvsp.html
[2]http://www.ic.unicamp.br/ cid/Problem-instances/VSP.html#VSP
[3]http://www.optsicom.es/maxcut/#instances
[4]http://www.optsicom.es/vs

Table 1: Parameter setting of the PR-VSP algorithm

Parameters	Section	Description	Value
p	2.3	$RefSet$ size	20
τ	2.3	coefficient used in the distance threshold	0.3
α	2.4	coefficient used in the tabu tenure	1.6
β	2.4	coefficient used in the iteration cutoff	2.4
ρ	2.4	coefficient for the perturbation method	rand(0.05,0.25)

type are widely observed in the Internet, the World Wide Web, citation networks and some social networks. This set of instances was tested in Sánchez-Oro et al. (2014).

- **Erdos-Renyi benchmarks:** This set[5] contains 95 random instances with $100 \leq |V| \leq 1000$ and each pair of vertices connected with a probability randomly chosen from $[0.2, 1.0]$. This set of instances was tested in Sánchez-Oro et al. (2014).

Our PR-VSP algorithm was programmed in C++ and compiled using GNU g++ on a Xeon E5440 (2.83 GHz CPU and 8 GB of RAM). The following time limits were used as stopping conditions of our experiments: 1 second for the traditional benchmarks, 3600 seconds for the Hermberg and Rendl benchmarks and 10 seconds for both the Barabasi-Albert and Erdos-Renyi benchmarks. Given the stochastic nature of the PR-VSP algorithm, we run PR-VSP to solve each problem instance 100 times independently and report computational statistics based on the outcomes of the 100 runs.

3.2. Parameter setting

Table 1 shows the parameter setting of the PR-VSP algorithm used for our experiments. To identify the adopted parameter values, we conducted a parameter sensitivity analysis on a set of 20 representative instances by comparing different values for each parameter: $p \in \{10, 15, 20, 25, 30\}$, $\alpha \in \{0.4, 0.8, 1.2, 1.6, 2.0\}$, $\beta \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$, $\rho \in \{rand(0.05, 0.20), rand(0.05, 0.25), rand(0.10, 0.25), rand(0.15, 0.25), rand(0.15, 0.30)\}$ and $\gamma \in \{0.2, 0.25, 0.3, 0.35, 0.4\}$. By varying the values of one parameter and keeping the values of the other parameters unchanged, we ran the PR-VSP algorithm 20 times to solve each chosen instance and recorded the average solution values. Hence, we obtained a table for each parameter where the columns represent different values for this parameter and the rows represent the average solution values for each instance. Furthermore, we employed Friedman statistical tests to verify if different values for a specific parameter present statistical differences.

Experimental results indicated that varying values of the parameters p, β, ρ and γ present no significant differences with p-values of 0.7925, 0.5374, 0.4147 and 0.8769, respectively. This means that the algorithm is not sensitive to these four parameters. However, the p-value of 0.0007 for the parameter α indicates

[5]http://www.optsicom.es/vs

Table 2: Post-hoc statistical tests for the parameter α

α	0.4	0.8	1.2	1.6
0.8	0.8853			
1.2	0.0474	0.1612		
1.6	0.0015	0.0019	0.3327	
2.0	0.0231	0.1291	0.5298	0.6241

that the algorithm is sensitive to the tabu tenure. Furthermore, we conducted a post-hoc analysis to check statistical differences between each pair of α values and showed the results in Table 2. As it can be seen in Table 2, four pairs of values present significant differences with a p-value < 0.05, among which two pairs are related to the setting $\alpha = 1.6$. In order to choose the best parameter setting, we also evaluated the number of best solutions achieved by each setting as a secondary criterion. The results showed that the setting $\alpha = 1.6$ obtains the best solution for 18 out of the 20 tested instances and performed the best among all the settings. In conclusion, this experiment reveals the rationality of the chosen parameter setting of Table 1.

3.3. Reference algorithms

For the purpose of our comparative study, we used the following state-of-the-art algorithms as our references.

- Breakout local search (BLS) (Benlic & Hao, 2013) is a heuristic algorithm which reported results on the 104 traditional benchmarks as well as the 71 Hermberg and Rendl benchmarks. Like our PR-VSP algorithm, BLS was written in C++ and compiled with GNU g++ under GNU/Linux running on an Intel Xeon E5440 (2.83 GHz and 2 GB of RAM). The stopping condition was a maximum running time of 10 seconds for the 104 traditional benchmarks and 3600 seconds for the 71 Hermberg and Rendl benchmarks.

- General variable neighborhood search (GVNS) (Sánchez-Oro et al., 2014) is a heuristic algorithm which reports results on the 104 traditional benchmarks, the 95 Barabasi-Albert benchmarks and the 95 Erdos-Renyi benchmarks. GVNS was implemented in Java SE7 and the results were obtained on a computer with an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB of RAM. The stopping condition used was a maximum running time of 5 seconds for the 104 traditional benchmarks and 1800 seconds for the other benchmarks.

- B-S (de Souza & Balas, 2005) is a branch-and-cut exact algorithm based on the results of an in-depth polyhedral study. Computational reports were reported on the 104 traditional benchmarks on a Pentium 4 computer (2.5 GHz and 2 GB of RAM) with a time limit of 1800 seconds.

- NDHYBRID is the best hybrid algorithm proposed in (de Souza & Cavalcante, 2011), which combines a branch-and-cut algorithm with a relax-and-cut algorithm as the pre-processing phase. The NDHYBRID algorithm reported results on 40 out of the 104 traditional instances and additional 7 MIPLIB instances (Borndorfer et al., 1998). These results were obtained on a Pentium 4 computer (2.66 GHz and 1 GB of RAM) with a stopping condition of 1800 seconds. Unfortunately, the MIPLIB instances tested are no more available and thus we focus our comparison on the 40 tested traditional instances.

- B-M (Biha & Meurs, 2011) is an exact approach which applies the general CPLEX 9.0 solver to a mixed-integer program. The results on the 104 traditional benchmarks were obtained on a Pentium M740 computer with 1.73 GHz and 1 GB of RAM. The stopping condition was not explicitly indicated in the paper.

Given that the compared algorithms (except BLS) were run on computing platforms which are different from our computer, it is difficult to make a fair comparison of the computing times. For this reason, we focused our comparisons on the solution quality criterion while providing the timing information only for indicative purposes. To make the time information somewhat meaningful, we used the CPU performance measurement suits from the well-known SPEC (https://www.spec.org/benchmarks.html) to normalize the computing times of the compared algorithms with our machine as the reference. As such, we multiplied the computing times reported by GVNS, B-S, NDHYBRID and B-M by 1.2, 0.8, 0.8 and 0.6 respectively. It is important to note that the normalized ratios from SPEC do not ensure an exact run time conversion among the compared algorithms, given that the run time of an algorithm also depends on multiple factors such as the programming language, data structures, and compiler options. Consequently, the timing information is to be interpreted with caution.

3.4. Computational results and comparisons

Table 3 shows the computational results on the 104 traditional instances obtained by our PR-VSP algorithm along with the results of four reference algorithms: breakout local search (BLS) (Benlic & Hao, 2013), general variable neighborhood search (GVNS) (Sánchez-Oro et al., 2014) and the three exact algorithms presented in (de Souza & Balas, 2005; de Souza & Cavalcante, 2011; Biha & Meurs, 2011). Since this set of benchmark instances have known optimal solutions, we report the number of instances for which the optimal solutions are obtained by each algorithm and the computational time. For the two heuristics (PR-VSP and BLS), we indicate the best time, the average time and the worst time in seconds. From Table 3, we find that our algorithm is able to reach the optimal solutions for all the 104 instances, with a worst time of 0.82 seconds and an average time of 0.03 seconds, which is the shortest among all the compared algorithms. Among the three exact algorithm, only B-M (Biha & Meurs, 2011)

Table 3: Computational results of the PR-VSP algorithm on the set of 104 small traditional instances in comparison with four reference algorithms: BLS (Benlic & Hao, 2013), GVNS (Sánchez-Oro et al., 2014), B-S (de Souza & Balas, 2005), NDHYBRID (de Souza & Cavalcante, 2011) and B-M (Biha & Meurs, 2011)

Algorithms	t_{avg}	t_{best}	t_{worst}	#solved instances
PR-VSP	0.03	0.00	0.82	104/104
BLS	0.08	0.00	3.06	104/104
GVNS	4.81	0.55	10.81	104/104
B-S	62.18	-	1131.60	97/104
NDHYBRID	63.11	-	592.13	37/40
B-M	140.28	-	9783.08	104/104

was able to solve all the 104 instances with long run time up to 9783.08 seconds, while B-S (de Souza & Balas, 2005) and NDHYBRID (de Souza & Cavalcante, 2011) failed to solve 7 out of 104 instances and 3 out of 40 instances respectively. According to these results, we conclude that these 104 traditional instances do not represent any challenge any more.

Table 4 is dedicated to the set of 71 Hermberg and Rendl benchmark instances and presents the comparative results between the PR-VSP algorithm and the state-of-the-art BLS algorithm. The second column (f_{prev}) indicates the current best known results reported in the literature. The results of PR-VSP and BLS are respectively shown in columns 3-5 and columns 6-8 in terms of the best solution value *Best*, the average solution value *Avg* and the average time *Time* to reach *Best*. To make a fair comparison, we reran BLS on our computer under the same time limit as our PR-VSP algorithm. From Table 4, we observe that PR-VSP is able to find new best solutions (displayed in bold) for 22 out of 71 instances and fails to reach the best known results for only one instance (G46). Moreover, PR-VSP obtains better, equal and worse average solution values relative to the BLS algorithm for 45, 14 and 12 instances, respectively, demonstrating its competitiveness compared to BLS in terms of solution quality. Finally, the computational time taken by PR-VSP to reach better solutions is competitive with the time taken by BLS.

Tables 5 and 6 compare the PR-VSP and GVNS algorithms on 90 Barabasi-Albert instances and 90 Erdos-Renyi instances, respectively. For the PR-VSP algorithm, we report the best solution value *Best*, average solution value *Avg* and the computational time *Time* to reach *Best* obtained for each instance. The results of the GVNS algorithm are taken from Sánchez-Oro et al. (2014). Since the two compared algorithms are implemented in different languages and have been tested on different computing platforms, the timing information was provided only for indicative purposes. As shown in Tables 5 and 6, PR-VSP consistently attains better solutions (shown in bold) than GVNS for 26 Barabasi-Albert instances and 20 Erdos-Renyi instances, respectively. For the other instances, our algorithm matches the best solution values found by the GVNS algorithm. In particular, the computational time of PR-VSP is 50 times shorter than that of GVNS on average.

Table 4: Computational results of the PR-VSP algorithm on the set of 71 Hermberg and Rendl instances in comparison with the state-of-the-art BLS algorithm (Benlic & Hao, 2013)

Instances	f_{prev}	PR-VSP			BLS		
		Best	*Avg*	Time	*Best*	*Avg*	Time
G1	257	257	257	0.84	257	257	8.23
G2	257	257	257	0.38	257	257	7.49
G3	257	257	257	1.43	257	257.05	76.35
G4	363	363	363	11.54	363	363.5	1735.65
G5	257	257	257	5.18	257	257	180.59
G6	257	257	257	0.41	257	257	7
G7	257	257	257	0.63	257	257	5.78
G8	257	257	257	1.92	257	257	153.27
G9	257	257	257	1.37	257	257	29.89
G10	257	257	257	3.56	257	257	220.92
G11	16	16	16	0.15	16	16	0.14
G12	32	32	32	0.08	32	32	0.05
G13	45	45	46.8	69.75	45	45	5.02
G14	146	146	146.3	386.15	146	146	1009.69
G15	144	144	144	12.98	144	144	13.83
G16	144	144	144	11.29	144	144	8.38
G17	144	144	144	55.89	144	144	54.88
G18	146	146	146.1	184.52	146	146	632.42
G19	144	144	144	8.97	144	144	19.47
G20	144	144	144	14.73	144	144	16.24
G21	144	144	144.1	67.01	144	144	16.08
G22	588	**587**	587	826.47	588	588.4	1023.94
G23	590	590	590	10.06	590	590.4	1342.36
G24	589	**587**	587.9	1228.16	589	589.5	1384.47
G25	589	**588**	588.3	1515.4	589	589.2	841.67
G26	587	587	587	671.46	588	588.15	1005.26
G27	820	**818**	818.7	815.85	820	820.05	798.99
G28	822	**821**	821.7	996.89	822	822.95	163.71
G29	820	**819**	819	1246.36	820	820.75	1922.14
G30	821	**820**	820.6	1716.18	821	821.75	1041.71
G31	819	819	819	976.61	819	819.65	1771.11
G32	40	40	40	0.44	40	40	0.66
G33	50	50	50	0.26	50	50	0.2
G34	80	80	82	0.21	80	82	0.14
G35	436	**435**	435.2	2025.4	436	436.35	1696.19
G36	441	**440**	440.4	1105.17	441	442.05	1302.49
G37	435	**434**	434.7	2307.84	435	438.2	2211.1
G38	439	439	439	1010.22	439	440.3	2156.96
G39	436	**435**	435.3	1415.07	436	437.8	1843.45
G40	440	440	440.4	1129.67	440	442.1	2365.89
G41	435	**434**	434.5	1160.87	435	437.05	1400.01
G42	439	**438**	438.8	650	439	440.8	2080.13
G43	411	411	411	5.28	411	411	11.52
G44	411	411	411	1.86	411	411	247.13
G45	410	410	410	4.04	410	410	53.78
G46	411	*412*	412	0.96	412	412	3.92
G47	411	411	411	17.88	411	411.95	0.62
G48	100	100	104	0.49	100	102	0.82
G49	60	60	60	1.25	60	60	0.52
G50	50	50	50	1.02	50	50	0.99
G51	224	224	224	38.18	224	224	59.11
G52	223	223	223.4	383.27	223	223.25	1140.31
G53	221	221	221.2	187.11	221	221.35	626.08
G54	219	219	219	18.14	219	219	32.88
G55	995	**979**	987	2752.88	997	1006.95	3170.29
G56	999	**972**	987.7	3345.15	999	1010.4	2357.58
G57	100	100	110	8.11	100	100	1.25
G58	1109	**1085**	1101	3352.99	1109	1133.35	3433.27
G59	1105	**1088**	1102.2	776.48	1105	1127.1	3396
G60	1376	**1354**	1372	3375.54	1386	1397.15	3343.69
G61	1385	**1350**	1368.2	3561.93	1385	1397.8	2653.87
G62	140	140	146	1.67	140	149	43.09
G63	1575	**1546**	1560.4	3321.63	1575	1596.85	2139.52
G64	1582	**1549**	1566.6	3363.47	1582	1602.6	3205.26
G65	160	160	164	7.72	160	160	34.39
G66	180	180	184	39.73	180	181	35.56
G67	194	194	197	782.94	194	196.7	411.21
G70	605	**320**	328.1	2977.13	609	633.25	2503.34
G72	194	194	197.5	487.76	194	195.5	643.86
G77	200	200	219.6	136.63	200	206.2	632.68
G81	200	200	220	4.58	200	213.85	39.27
Better		22					

Table 4 – continued from previous page

Instances	f_{prev}	PR-VSP			BLS		
		Best	*Avg*	Time	*Best*	*Avg*	Time
Equal		48					
Worse		1					

Table 5: Computational results of the PR-VSP algorithm on the set of 95 Barabasi-Albert instances in comparison with the state-of-the-art GVNS algorithm (Sánchez-Oro et al., 2014)

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
barabasi_albert_1(100,65)	43	43	0.02	43	5.13
barabasi_albert_1(1000,878)	564	564	3.24	564	93.57
barabasi_albert_1(150,137)	86	86	0.04	86	7.64
barabasi_albert_1(200,175)	112	112	0.06	112	10.11
barabasi_albert_1(250,146)	99	99	0.26	99	13.01
barabasi_albert_1(300,255)	160	160	0.19	160	16.01
barabasi_albert_1(350,320)	198	198	0.23	198	17.98
barabasi_albert_1(400,376)	234	234	0.24	234	20.86
barabasi_albert_1(450,326)	218	218	0.55	218	23.45
barabasi_albert_1(500,277)	204	204	0.51	204	25.16
barabasi_albert_1(550,499)	314	314	1	314	33.41
barabasi_albert_1(600,541)	**348**	348	1.1	349	32.74
barabasi_albert_1(650,465)	320	320	0.71	320	45.84
barabasi_albert_1(700,649)	**409**	409	1.54	415	40.76
barabasi_albert_1(750,422)	303	303	1.11	303	59.74
barabasi_albert_1(800,627)	418	418	1.37	418	59.07
barabasi_albert_1(850,619)	418	418	2.86	418	76.26
barabasi_albert_1(900,817)	**522**	522	4.63	527	59.72
barabasi_albert_1(950,626)	**442**	442	2	444	57.83
barabasi_albert_2(100,69)	45	45	0.02	45	5.05
barabasi_albert_2(1000,856)	556	556	4.64	556	100.28
barabasi_albert_2(150,94)	65	65	0.04	65	7.66
barabasi_albert_2(200,161)	105	105	0.09	105	10.17
barabasi_albert_2(250,235)	147	147	0.12	147	12.5
barabasi_albert_2(300,220)	**147**	147	0.17	148	15.11
barabasi_albert_2(350,182)	129	129	0.16	129	21.29
barabasi_albert_2(400,227)	**164**	164	0.32	165	23.78
barabasi_albert_2(450,392)	252	252	0.78	252	25.52
barabasi_albert_2(500,288)	205	205	0.46	205	36.97
barabasi_albert_2(550,355)	**242**	242	0.74	247	31.19
barabasi_albert_2(600,520)	335	335	0.7	335	35.86
barabasi_albert_2(650,485)	327	327	1.02	327	41.45
barabasi_albert_2(700,545)	368	368	1.85	368	47.1
barabasi_albert_2(750,395)	**285**	285	0.66	293	61.85
barabasi_albert_2(800,617)	**416**	416	1.14	419	54.16
barabasi_albert_2(850,739)	478	478	4.07	478	53.72
barabasi_albert_2(900,576)	**404**	404	1.04	411	49.2
barabasi_albert_2(950,744)	**501**	501	4.72	505	91.86
barabasi_albert_3(100,64)	43	43	0.02	43	5.1
barabasi_albert_3(1000,601)	430	430	2.34	430	72.14
barabasi_albert_3(150,129)	83	83	0.04	83	7.62
barabasi_albert_3(200,111)	81	81	0.07	81	10.66
barabasi_albert_3(250,191)	124	124	0.15	124	13.26
barabasi_albert_3(300,260)	159	159	0.11	159	15.56
barabasi_albert_3(350,251)	166	166	0.3	166	17.98
barabasi_albert_3(400,284)	**191**	191	0.21	193	22.92
barabasi_albert_3(450,243)	**177**	177	0.25	179	24.86
barabasi_albert_3(500,273)	200	200	0.47	200	25.12
barabasi_albert_3(550,294)	217	217	0.29	217	35.87
barabasi_albert_3(600,435)	293	293	0.82	293	35.01
barabasi_albert_3(650,642)	387	387	0.76	387	41.02
barabasi_albert_3(700,678)	417	417	0.83	418	37.65
barabasi_albert_3(750,643)	416	416	0.74	416	38.4
barabasi_albert_3(800,595)	**399**	399	5.16	409	59.2
barabasi_albert_3(850,693)	**453**	453	2.3	458	65.65
barabasi_albert_3(900,851)	535	535	1.81	535	54.22
barabasi_albert_3(950,553)	**398**	398	1.48	401	62.09
barabasi_albert_4(100,87)	51	51	0.02	51	5.1
barabasi_albert_4(1000,509)	**381**	381	3.31	391	78.21

Table 5 – continued from previous page

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
barabasi_albert_4(150,111)	71	71	0.05	71	7.55
barabasi_albert_4(200,197)	127	127	0.07	127	10.11
barabasi_albert_4(250,133)	98	98	0.16	98	13.23
barabasi_albert_4(300,205)	139	139	0.23	139	17.08
barabasi_albert_4(350,294)	188	188	0.27	188	17.53
barabasi_albert_4(400,350)	225	225	0.22	225	22.92
barabasi_albert_4(450,229)	165	165	0.19	165	26.05
barabasi_albert_4(500,496)	305	305	0.3	305	26.78
barabasi_albert_4(550,347)	245	245	0.46	245	31.13
barabasi_albert_4(600,305)	226	226	0.43	226	31.05
barabasi_albert_4(650,535)	347	347	0.59	347	36.09
barabasi_albert_4(700,621)	395	395	1.18	395	45.1
barabasi_albert_4(750,722)	**447**	447	1	453	42.79
barabasi_albert_4(800,750)	477	477	1.08	477	59.9
barabasi_albert_4(850,646)	434	434	1.44	434	72.54
barabasi_albert_4(900,768)	**504**	504	3.97	510	68.77
barabasi_albert_4(950,758)	507	507	0.99	507	89.51
barabasi_albert_5(100,89)	55	55	0.02	55	5.05
barabasi_albert_5(1000,578)	413	413	1.19	413	74.64
barabasi_albert_5(150,103)	67	67	0.05	67	7.64
barabasi_albert_5(200,199)	132	132	0.05	132	10.15
barabasi_albert_5(250,132)	94	94	0.18	94	13.07
barabasi_albert_5(300,211)	139	139	0.13	139	16.11
barabasi_albert_5(350,249)	**164**	164	0.16	168	18.01
barabasi_albert_5(400,233)	162	162	0.35	162	23.82
barabasi_albert_5(450,424)	**269**	269	0.38	270	25.17
barabasi_albert_5(500,408)	270	270	1.48	270	27.48
barabasi_albert_5(550,495)	317	317	0.48	317	31.76
barabasi_albert_5(600,475)	316	316	0.73	316	30.97
barabasi_albert_5(650,434)	**298**	298	0.68	304	47.87
barabasi_albert_5(700,501)	**341**	341	0.67	346	35.7
barabasi_albert_5(750,744)	453	453	1.71	453	43.28
barabasi_albert_5(800,663)	432	432	0.75	432	49.66
barabasi_albert_5(850,635)	**430**	430	3.16	433	71.7
barabasi_albert_5(900,662)	**446**	446	1.27	452	88.55
barabasi_albert_5(950,818)	534	534	2.48	534	83.49
Better	25				
Equal	70				
Worse	0				

Table 6: Computational results of the PR-VSP algorithm on the set of 95 Erdos-Renyi instances in comparison with the state-of-the-art GVNS algorithm (Sánchez-Oro et al., 2014).

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
erdos_renyi_1(100,0.89)	82	82	0.02	82	5.004
erdos_renyi_1(1000,0.27)	333	333	0.4	333	66.929
erdos_renyi_1(150,0.86)	118	118	0.06	118	7.531
erdos_renyi_1(200,0.82)	147	147	0.07	147	10.202
erdos_renyi_1(250,0.89)	205	205	0.1	205	12.583
erdos_renyi_1(300,0.34)	99	99	0.06	99	16.779
erdos_renyi_1(350,0.32)	116	116	0.07	116	20.186
erdos_renyi_1(400,0.79)	290	290	0.34	290	20.517
erdos_renyi_1(450,0.25)	149	149	0.09	149	22.897
erdos_renyi_1(500,0.95)	454	454	0.47	454	25.097
erdos_renyi_1(550,0.64)	316	316	0.45	316	36.505
erdos_renyi_1(600,0.59)	**316**	316	0.56	318	39.628
erdos_renyi_1(650,0.24)	216	216	0.17	216	38.103
erdos_renyi_1(700,0.41)	**243**	243	0.46	254	57.626
erdos_renyi_1(750,0.57)	**394**	394	1.05	401	54.65
erdos_renyi_1(800,0.31)	266	266	0.3	266	76.686
erdos_renyi_1(850,0.91)	746	746	1.77	746	44.4
erdos_renyi_1(900,0.37)	299	299	0.43	299	55.387
erdos_renyi_1(950,0.81)	733	733	2.39	733	54.825
erdos_renyi_2(100,0.12)	28	28	0.01	28	5.095
erdos_renyi_2(1000,0.30)	333	333	0.41	333	69.413
erdos_renyi_2(150,0.51)	60	60	0.03	60	7.87
erdos_renyi_2(200,0.23)	65	65	0.38	65	10.825

Table 6 – continued from previous page

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
erdos_renyi_2(250,0.81)	183	183	0.1	183	12.812
erdos_renyi_2(300,0.52)	132	132	0.09	132	15.142
erdos_renyi_2(350,0.19)	115	115	0.05	115	20.724
erdos_renyi_2(400,0.40)	133	133	0.13	133	25.606
erdos_renyi_2(450,0.10)	**144**	144	1.04	145	29.39
erdos_renyi_2(500,0.05)	**153**	153	4.03	155	34.925
erdos_renyi_2(550,0.33)	183	183	0.15	183	36.158
erdos_renyi_2(600,0.21)	**198**	198.1	0.19	199	44.549
erdos_renyi_2(650,0.36)	216	216	0.24	216	46.489
erdos_renyi_2(700,0.49)	300	300	0.45	300	52.715
erdos_renyi_2(750,0.94)	678	678	0.95	678	38.341
erdos_renyi_2(800,0.36)	266	266	0.36	266	41.346
erdos_renyi_2(850,0.64)	**506**	506	3.06	511	61.056
erdos_renyi_2(900,0.61)	**507**	507	0.92	511	79.593
erdos_renyi_2(950,0.83)	**754**	754	1.62	755	50.552
erdos_renyi_3(100,0.78)	61	61	0.02	61	5.009
erdos_renyi_3(1000,0.92)	891	891	6.2	891	52.155
erdos_renyi_3(150,0.38)	49	49	0.03	49	7.88
erdos_renyi_3(200,0.35)	66	66	0.03	66	10.753
erdos_renyi_3(250,0.37)	83	83	0.05	83	14.101
erdos_renyi_3(300,0.25)	99	99	0.05	99	15.334
erdos_renyi_3(350,0.55)	161	161	0.14	161	18.249
erdos_renyi_3(400,0.11)	**129**	129	2.46	130	22.915
erdos_renyi_3(450,0.75)	309	309	0.41	309	22.976
erdos_renyi_3(500,0.50)	**211**	211	0.27	223	36.001
erdos_renyi_3(550,0.87)	452	452	0.57	452	27.84
erdos_renyi_3(600,0.25)	199	199	0.14	199	30.911
erdos_renyi_3(650,0.45)	246	246	0.28	246	46.143
erdos_renyi_3(700,0.44)	**265**	265	0.39	278	55.913
erdos_renyi_3(750,0.94)	676	676	0.86	676	38.087
erdos_renyi_3(800,0.61)	437	437	1.07	437	65.796
erdos_renyi_3(850,0.27)	283	283	0.29	283	84.293
erdos_renyi_3(900,0.81)	686	686	1.77	686	47.637
erdos_renyi_3(950,0.80)	726	726	1.53	726	55.15
erdos_renyi_4(100,0.32)	33	33	0.02	33	5.01
erdos_renyi_4(1000,0.55)	**507**	507	1.59	512	67.073
erdos_renyi_4(150,0.69)	89	89	0.05	89	7.6
erdos_renyi_4(200,0.61)	102	102	0.06	102	10.423
erdos_renyi_4(250,0.69)	153	153	0.1	153	12.757
erdos_renyi_4(300,0.35)	99	99	0.06	99	16.594
erdos_renyi_4(350,0.22)	**115**	115.4	0.06	116	18.767
erdos_renyi_4(400,0.86)	307	307	0.23	307	20.179
erdos_renyi_4(450,0.94)	407	407	0.38	407	22.679
erdos_renyi_4(500,0.75)	**343**	343	0.65	344	27.087
erdos_renyi_4(550,0.83)	432	432	0.51	432	29.448
erdos_renyi_4(600,0.76)	425	425	0.74	425	35.145
erdos_renyi_4(650,0.59)	**347**	347	1.13	348	33.057
erdos_renyi_4(700,0.62)	**390**	390	0.71	397	39.808
erdos_renyi_4(750,0.57)	380	380	1.1	380	55.152
erdos_renyi_4(800,0.98)	765	765	1.14	765	40.252
erdos_renyi_4(850,0.74)	582	582	3.57	592	52.937
erdos_renyi_4(900,0.35)	299	299	0.4	299	57.307
erdos_renyi_4(950,0.45)	371	371	0.67	371	67.883
erdos_renyi_5(100,0.71)	58	58	0.02	58	5.004
erdos_renyi_5(1000,0.86)	821	821	2.7	821	57.026
erdos_renyi_5(150,0.07)	**40**	40	0.02	42	8.252
erdos_renyi_5(200,0.44)	69	69	0.04	69	10.815
erdos_renyi_5(250,0.68)	149	149	0.09	149	12.991
erdos_renyi_5(300,0.36)	99	99	0.07	99	16.772
erdos_renyi_5(350,0.55)	**170**	170	0.14	172	20.441
erdos_renyi_5(400,0.38)	133	133	0.12	133	25.181
erdos_renyi_5(450,0.25)	149	149	0.09	149	22.936
erdos_renyi_5(500,0.21)	165	165.1	0.34	165	30.532
erdos_renyi_5(550,0.60)	290	290	0.44	290	31.482
erdos_renyi_5(600,0.24)	199	199	0.15	199	30.702
erdos_renyi_5(650,0.65)	**386**	386	1.09	390	41.257
erdos_renyi_5(700,0.94)	633	633	1.01	633	35.291
erdos_renyi_5(750,0.70)	473	473	0.98	473	52.227
erdos_renyi_5(800,0.38)	266	266	0.72	266	42.812
erdos_renyi_5(850,0.33)	283	283	0.33	283	44.077
erdos_renyi_5(900,0.22)	299	299	0.29	299	47.898
erdos_renyi_5(950,0.29)	316	316	0.36	316	59.508
Better	20				

Table 6 – continued from previous page

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
Equal	75				
Worse	0				

3.5. Analysis

To complement the computational studies presented in the last section, we now provide an analysis of some key ingredients of the proposed PR-VSP algorithm to shed light on their impacts on the performance of the algorithm. As explained in Section 2, relative to the existing leading heuristics like Benlic & Hao (2013); Sánchez-Oro et al. (2014), PR-VSP includes two distinguishing features: a new local search operator (i.e., *swap-move*) and a dedicated path relinking procedure. In order to assess their contributions, we create two PR-VSP variants by disabling the *swap-move* operator (denoted by PR_non-swap) and the path relinking procedure (denoted by ITS). We compare PR-VSP with these two variants based on a selection of 31 representative instances.

For this experiment, we ran the two PR variants under the same condition as the PR-VSP algorithm. The results are summarized in Table 7 where we indicate the best solution value *Best*, the average solution value *Avg* and the average time *Time* to reach *Best* obtained by PR-VSP, PR_non-swap and ITS. The results of PR-VSP are directly extracted from Table 4. As shown in Table 7, PR-VSP dominates PR_non-swap and ITS since the two variants can only attain respectively 13 and 11 *Best* values reported by PR-VSP for the 31 instances. Moreover, PR-VSP performs better in terms of the average solution value, with an average of 680.4 against 691.76 for PR_non-swap and 685.82 for ITS. In addition, the computing time of PR-VSP remains competitive with those of PR_non-swap and ITS, while attaining solutions of higher quality.

This experiment demonstrates the effectiveness of the new *swap-move* operator and the dedicated relinking procedure to the performance of the PR-VSP algorithm.

4. Conclusion

We presented the first path relinking algorithm for solving the NP-hard vertex separator problem. The proposed PR-VSP algorithm integrates specially a new swap operator in its local improvement procedure and a dedicated relinking procedure for path generations. The proposed algorithm was assessed on four sets of 365 benchmark instances with up to 20,000 vertices. Comparisons with the best performing algorithms in the literature showed that our algorithm competes very favorably with the reference methods. Specifically, it is able to find improved best solutions (new upper bounds) for 67 large instances and matches previously best known results for all but one instance. The experiments also indicated that the proposed algorithm is computationally effective.

As future work, we can consider another form of path relinking called Exterior Path Relinking (Glover, 2014), which offers the possibility of including,

Table 7: Comparative results on 31 instances between PR-VSP and two variants

Instances	PR-VSP			PR_non-swap			ITS		
	Best	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*
G1	257	257	0.84	257	257	0.25	257	257	0.34
G10	257	257	3.56	257	257	1.21	257	257	5.16
G14	146	146.3	386.15	146	146.65	502.14	147	147	280.09
G21	144	144.1	67.01	144	144.2	115.4	144	144.95	619.74
G22	587	587	826.47	588	588.65	1520.39	588	589.66	1745.23
G23	590	590	10.06	590	590	50.57	590	590.95	315.59
G24	587	587.9	1228.16	588	588.35	1821.29	589	590.64	1564.18
G25	588	588.3	1515.4	589	589.5	1747.17	589	589.65	1573.66
G26	587	587	671.46	587	588.3	837.26	587	588.57	2552.6
G27	818	818.7	815.85	819	819.95	967.38	820	820.75	2136.41
G28	821	821.7	996.89	822	822.4	1372.29	822	822.95	1947.28
G29	819	819	1246.36	820	820.3	724.32	819	820.3	2841.23
G30	820	820.6	1716.18	820	820.6	1936.48	820	820.8	1901.36
G35	435	435.2	2025.4	435	435.65	2825.71	435	436.65	1576.71
G36	440	440.4	1105.17	440	440.95	1247.53	441	441.6	1647.73
G37	434	434.7	2307.84	435	436.7	539.92	435	436.7	1969.17
G38	439	439	1010.22	439	441.1	1828.57	440	441.5	1975.54
G39	435	435.3	1415.07	436	438.37	1521.74	437	438.95	1724.79
G40	440	440.4	1129.67	440	441.95	1019.46	440	441.95	2104.56
G41	434	434.5	1160.87	435	438.39	1437.12	435	441.15	2017.85
G47	411	411	17.88	411	411.3	135.58	411	412.3	887.64
G51	224	224	38.18	224	224.4	121.52	224	225.8	472.4
G55	979	987	2752.88	989	992.8	3301.25	984	990.45	3017.85
G56	972	987.7	3345.15	989	997.05	3470.67	976	998.12	3214.04
G58	1085	1101	3352.99	1092	1105.78	3102.21	1090	1103.72	2974.58
G59	1088	1102.2	776.48	1090	1109.13	1034.11	1094	1101.45	2457.45
G60	1354	1372	3375.54	1369	1385.8	1509.75	1359	1375.8	2434.87
G61	1350	1368.2	3561.93	1380	1398.12	2087.28	1356	1368.3	2641.79
G63	1546	1560.4	3321.63	1552	1565.4	2935.84	1563	1575.55	3017.75
G64	1549	1566.6	3363.47	1553	1564.36	3157.48	1559	1579.65	2974.47
G70	320	328.1	2977.13	505	584.26	3015.65	401	410.6	2748.64
AVG	676.00	680.4	1500.71	685.19	691.76	1480.24	680.94	685.82	1849.7
best/total	31/31			13/31			11/31		

during the relinking process, characteristics other than those of the guiding solution. It would also be interesting to verify the merit of the newly defined swap operator for other graph partition problems.

Acknowledgements

We are grateful to the reviewers and the Editor whose comments have helped us to improve the paper. This work was partially supported by the National Natural Science Foundation of China (Grant No. 71501157).

References

Ashcraft, C. C., & Liu, J. W. H. (1994). A partition improvement algorithm for generalized nested dissection. Technical Report BCSTECH-94-020, Boeing Computer Services, Seattle, WA.

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

Benlic, U., & Hao, J.-K. (2013). Breakout local search for the vertex separator problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* IJCAI '13 (pp. 461–467). AAAI Press.

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

Borndörfer, R., Ferreira, C. E., & Martin, A. (1998). Decomposing matrices into blocks. *SIAM Journal on Optimization* 9(1), 236–269.

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

Chen, Y., Hao, J.-K., & Glover, F. (2016). An evolutionary path relinking approach for the quadratic multiple knapsack problem. *Knowledge-Based Systems*, *92*, 23–34.

Costa, W.E., Goldbarg, M.C., & Goldbarg, E.G. (2012). Hybridizing VNS and path-relinking on a particle swarm framework to minimize total flowtime. *Expert Systems with Applications* 39(18), 13118–13126.

Evrendilek, C. (2008). Vertex separators for partitioning a graph. *Sensors*, *8*, 635–657.

Feige, U., Hajiaghayi, M., & Lee, J. R. (2008). Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, *38*, 629–657.

Fiduccia, C.M., & Mattheyses, R.M. (1982). A linear-time heuristic for improving network partitions. *In 19th IEEE Conference on Design Automation*, 175–181.

Fu, B., & Chen, Z. (2006). Sublinear time width-bounded separators and their application to the protein side-chain packing problem. In *Algorithmic Aspects in Information and Management* (pp. 149–160). Springer.

Fukuyama, J. (2006). NP-completeness of the planar separator problems. *Journal of Graph Algorithms and Applications*, *10*, 317–328.

George, A., & Liu, J. W. H. (1981). Computer Solution of Large Sparse Positive Definite Systems. *Prentice-Hall, Englewood Cliffs, NJ.*

Glover, F., & Laguna, M. (1997). *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers.

Glover, F. (1998). A template for scatter search and path relinking. *Lecture Notes in Computer Science*, *1363*, 13–54.

Glover F. (2014) Exterior path relinking for zero-one optimization. *International Journal of Applied Metaheuristic Computing* 5(3): 1–8.

González, M. A., Oddi, A., Rasconi, R., & Varela, R. (2015). Scatter search with path relinking for the job shop with time lags and setup times. *Computers & Operations Research*, 60, 37–54.

Hager, W. W., & Hungerford, J. T. (2015). Continuous quadratic programming formulations of optimization problems on graphs. *European Journal of Operational Research*, *240*, 328–337.

Lai, X., & Hao, J.-K. (2015). Path relinking for the fixed spectrum frequency assignment problem. *Expert Systems with Applications*, *42*, 4755–4767.

Leighton, F. T. (1983). *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*. MIT press.

Lipton, R. J., & Tarjan, R. E. (1979). A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, *36*, 177–189.

Martins de Oliveira, R., Nogueira Lorena, L.A., Chaves, A.A., & Mauri, G.R. (2014). Hybrid heuristics based on column generation with path-relinking for clustering problems. *Expert Systems with Applications* 41(11), 5277–5284.

Mestria, M., Ochi, L.S., & Martins, S.D.L. (2013). GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem. *Computers & Operations Research*, 40(12), 3218–3229.

Parejo, J.A., Segura, S., Fernandez, P., & Ruiz-Cortés, A. (2014). QoS-aware web services composition using GRASP with Path Relinking. *Expert Systems with Applications* 41(9), 4211–4223.

Peng, B., Lü, Z., & Cheng, T. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, *53*, 154–164.

Sánchez-Oro, J., Mladenović, N., & Duarte, A. (2014). General variable neighborhood search for computing graph separators. *Optimization Letters*, (pp. 1–21).

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

de Souza, C. C., & Cavalcante, V. F. (2011). Exact algorithms for the vertex separator problem in graphs. *Networks*, *57*, 212–230.

Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flow shop problem. *Omega* 38(1-2), 57–67.

Wang, Y., Lü, Z., Glover, F., & Hao, J.-K. (2012). Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, *223*, 595–604.

Wang, Y., Wu, Q., & Glover, F. (2017). Effective metaheuristic algorithms for the minimum differential dispersion problem. *European Journal of Operational Research* 258(3), 829–843.

Wang, Y., & Punnen, A.P. (2017). The Boolean quadratic programming problem with generalized upper bound constraints. *Computers & Operations Research* 77: 1–10.

Zeng, R., Basseur, M., & Hao, J.-K. (2013). Solving bi-objective flow shop problem with hybrid path relinking algorithm. *Applied Soft Computing* 13(10), 4118–4132.