A Hybrid Breakout Local Search and Reinforcement Learning Approach to the Vertex Separator Problem

Una Benlic, Michael G. Epitropakis, Edmund K. Burke

PII: S0377-2217(17)30058-9
DOI: 10.1016/j.ejor.2017.01.023
Reference: EOR 14204

To appear in: *European Journal of Operational Research*

Received date: 14 July 2016
Revised date: 29 November 2016
Accepted date: 12 January 2017

Please cite this article as: Una Benlic, Michael G. Epitropakis, Edmund K. Burke, A Hybrid Breakout Local Search and Reinforcement Learning Approach to the Vertex Separator Problem, *European Journal of Operational Research* (2017), doi: 10.1016/j.ejor.2017.01.023

Highlights

- Improved motivation of the proposed method highlighting the main contributions;
- A highlight of the recent and important applications of the Vertex Separator Problem;
- Additional and relevant references;
- Improved and simplified presentation of the algorithm;
- Additional results showing the performance of the approach in terms of time and solution quality;

A Hybrid Breakout Local Search and Reinforcement Learning Approach to the Vertex Separator Problem

Una Benlic[a], Michael G. Epitropakis*[b], Edmund K. Burke[a]

[a] *School of Electronic Engineering and Computer Science, Queen Mary University of London, London, email: {u.benlic,e.burke}@qmul.ac.uk*
[b] *Data Science Institute, Department of Management Science, Lancaster Univeristy Management School, Lancaster University. email: m.epitropakis@lancaster.ac.uk*

Abstract

The Vertex Separator Problem (VSP) is an NP-hard problem which arises from several important domains and applications. In this paper, we present an improved Breakout Local Search for VSP (named BLS-RLE). The distinguishing feature of BLS-RLE is a new parameter control mechanism that draws upon ideas from reinforcement learning theory for an interdependent decision on the number and on the type of perturbation moves. The mechanism complies with the principle "intensification first, minimal diversification only if needed", and uses a dedicated sampling strategy for a rapid convergence towards a limited set of parameter values that appear to be the most convenient for the given state of search. Extensive experimental evaluations and statistical comparisons on a wide range of benchmark instances show significant improvement in performance of the proposed algorithm over the existing BLS algorithm for VSP. Indeed, out of the 422 tested instances, BLS-RLE was able to attain the best-known solution in 93.8% of the cases, which is around 20% higher compared to the existing BLS. In addition, we provide detailed analyses to evaluate the importance of the key elements of the proposed method and to justify the degree of diversification introduced during perturbation.

Keywords: Heuristics, Iterated local search, Vertex separator, Parameter control, Reinforcement learning

*Corresponding author

1. Introduction

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices with nonnegative weights $c_1, c_2, \ldots, c_n$ and let $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ be a set of unweighted edges. A vertex separator in a graph is a set of vertices whose removal breaks the graph into two non-empty disconnected components. More formally, the Vertex Separator Problem (VSP) is to find a partition of $\mathcal{V}$ into disjoint vertex subsets A, B, C with A and B non-empty, such that (i) there is no edge $(i, j) \in \mathcal{E}$, $i \in A$, $j \in B$; (ii) $\max\{|A|, |B|\} \leq b$, $1 \leq b \leq |\mathcal{V}|$ (b is the problem input) and; (iii) $\sum\{c_j : j \in C\}$ is minimized. A and B are called the 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. A separator C is balanced if $\max\{|A|, |B|\} \leq 2|\mathcal{V}|/3$.

The problem of finding minimal balanced separators is NP-hard [1]. It first arose in the context of Very Large Scale Integration (VLSI) design [2, 3], and has become popular in several other applications. For instance, in telecommunication networks, a separator determines the capacity and the brittleness of the network [2]. In bioinformatics and computational biology, separators in grid graphs provide a simplified representation of proteins [4]. The problem also finds its application in cyber security where it can be used to disconnect a largest connected component in a network to prevent a possible spread of an attack. A recent interesting application of VSP is in decoy routing, a mechanism capable of circumventing common network filtering strategies. The aim of decoy routing is to prevent nation-state level Internet censorship by having routers transfer traffic to blocked destinations. A decoy routing system needs to cover all paths to a large enough set of destinations such that it is economically or functionally infeasible for a warden to block these destinations. The problem of deploying a minimum number of decoy routers could be related to VSP [5, 6]. Generalization of the vertex separator problem also involves applications in distributed routing protocols [7]. But perhaps more importantly, finding small balanced separators is a major primitive for many graph algorithms, especially for those that are based on the principle of divide-and-conquer [8, 9, 10, 11, 12]. For instance, the author in [9] proposed a novel heuristic which partitions a graph via vertex separators so as to balance the workload amongst the processors and to minimize the communication overhead. Vertex separators have also been used for hypergraph partitioning, which has an impact on a wide variety of parallel and distributed computing applications [10].

Several exact algorithms have been proposed for solving VSP [13, 14,

15, 16]. These algorithms are able to find optimal results in a reasonable computing time (within 3 hours) for instances with up to 125 vertices, but fail to solve larger instances. Given the inherent computational complexity of the problem, approximation algorithms [17, 18, 19], heuristic [20, 21] and meta-heuristic methods have also been considered. The Breakout Local Search (BLS) algorithm [22], along with a Variable Neighborhood Search (VNS) [23], is among the first metaheuristic approaches applied to the above defined VSP formulation. BLS is a recent variation of the popular Iterated Local Search (ILS) metaheuristic [24] with a particular emphasis on the importance of the perturbation phase. In addition to VSP, BLS has been shown to provide competitive performance for several well-studied combinatorial problems including the quadratic assignment [25] and the maximum clique [26] problems.

Relying on the information related to the search state and progress, the basic idea of BLS is to use a perturbation mechanism to adaptively determine a suitable number and type of perturbation moves for the next round of the perturbation phase. Indeed, the existing BLS algorithm for VSP [22] determines both the number and the type of perturbation moves (random or directed moves) by exploiting information on a number of recently visited local optima stored in a hash table memory structure. Despite its success on several challenging combinatorial problems, this poses a limitation on its search dynamics since a more appropriate degree of diversification could be introduced if these two decisions are reactively determined in an interdependent manner.

The aim of this paper is twofold. Given the importance of VSP, our first objective is to enrich the VSP literature with a new self-adaptive approach, capable of finding high quality solutions with reasonable computing efforts for different types of problem landscapes. The other objective is to investigate how reinforcement learning techniques [27] can improve the reactive characteristic of a local search heuristic, i.e., a BLS algorithm. For these purposes, we introduce a hybrid between a Breakout Local Search and a Reinforcement Learning technique (denoted as BLS-RLE), which is an extension of the current BLS for VSP with a new parameter control mechanism based on reinforcement learning. The main novelty of the proposed algorithm is that it adaptively and interdependently determines the values for two important parameters: the number of perturbation moves l and the probability e of using a directed over a random perturbation type. Reinforcement learning techniques for parameter control have recently become a feature of the research agenda of the Evolutionary Computing community [28, 29, 30], and the Local Search community [31, 32]. The proposed

parameter control mechanism is modeled as the multi-armed bandit problem [33, 34], a reduced version of the reinforcement learning problem where learning is performed with respect to a single state. In a multi-armed bandit problem, the agent is repeatedly faced with a choice among different options, or actions. In our case, an action corresponds to the application of a parameter pair configuration (l, e) for the perturbation phase. After each action, the agent receives a numerical *reward* that captures the immediate impact on the search process. While the *reward* points out whether an action is good in an immediate sense, an *action value* is the total amount of reward an agent can expect to accumulate over the future. The objective of the multi-armed bandit problem is to maximize the expected action value over some time period. The originality and success of the proposed parameter controller lie in the two following key elements: (i) a dedicated strategy for sampling of configurations (l, e), which enables rapid convergence towards a limited set of actions (parameter pair settings) that appear to be the most convenient for the given state of search, and (ii) a reward function which complies with the principle "intensification first, minimal diversification only if needed" [35]. Furthermore, the proposed mechanism does not introduce a significant computational overhead compared to the existing BLS for VSP, while substantially improving the performance of BLS. Even though the objective of this work is to provide an effective approach to VSP, the proposed RL controller for BLS is general and can be considered for solving other important combinatorial problem. The corresponding code is available online for future use.

The existing BLS, as well as BLS-RLE, are able to solve to optimality all the instances from the current VSP benchmark [13] within less than a second [22]. To evaluate the performance of BLS-RLE with respect to BLS and several other algorithms from the ILS family, we thus use a new set of 426 challenging instances with different sizes and structures. A large number of these instances are motivated by several VSP applications including VLSI design, cyber security, and decoy routing. Extensive computational comparisons highlight the benefit of the proposed parameter control mechanism. Indeed, BLS-RLE significantly improves the performance of BLS and competes very favorably with several other variants of ILS. Briefly, out of the 422 considered instances, BLS-RLE was able to attain the best-known solution in 93.8% of the cases, which is around 20% higher compared to the BLS algorithm.

The rest of the paper is organized as follows. In Section 2, we provide a motivation and the general framework of the existing BLS method for combinatorial problems. Section 3 presents the improved BLS frame-

work and applies it to the vertex separator problem. Experimental results and extensive statistical comparisons are given in Section 4. Section 5 provides analyses of the problem landscape to evaluate the hardness of the used benchmark instances, and to justify the behavior of the proposed approach. This section further analyzes the importance of the key algorithmic components of BLS-RLE. Finally, conclusions are presented and discussed in Section 6.

2. Breakout Local Search

Iterated Local Search (ILS) [24] is a popular metaheuristic which has been used for tackling a large variety of hard combinatorial problems. Its basic idea is to iterate between a local search phase to intensify the search, and a perturbation phase to diversify the search. The degree of diversification introduced by an ILS method depends both on the number and on the type of moves applied for perturbation. For most ILS algorithms, these values are fixed throughout the search thus introducing a constant degree of diversification regardless of the search state. To cope with this limitation, a variation of ILS, named Breakout Local Search (BLS) [25, 26, 36], was recently proposed in the literature. BLS puts particular emphasis on the importance of the diversification phase. Based on relevant information on the search state, it adaptively determines an appropriate number of perturbation moves and selects between two or more types of perturbations of different intensities. More precisely, the idea behind BLS is to introduce weaker diversification as the search is visiting new unexplored solutions, and to gradually increase diversification as the number of repetitions of an already visited solution increases. BLS has previously been applied to the vertex separator problem and has been shown to outperform several variations of ILS on this problem [22].

Algorithm 1 shows the general scheme of BLS. To apply it to any combinatorial problem, four procedures need to be specified: *GenerateInitialSolution* generates an initial solution 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 one, and stops if such a solution is not found; *DetermineJumpMagnitude* determines the number l of perturbation moves ("jump magnitude"); *DeterminePerturbationType* selects the type t of perturbation moves among two or several alternatives of different diversification intensities. t is selected probabilistically, according to a probability e that is adaptively selected from a vector $\mathbf{e}$. In its simplest form with only two types of perturbation moves, e.g., directed and

Algorithm 1 General BLS framework

1: $s' \leftarrow GenerateInitialSolution$
2: $l \leftarrow l_0$ /*Initialise the number l of perturbation moves */
3: **while** stopping condition not reached **do**
4: $\quad s \leftarrow DescentBasedSearch(s')$
5: $\quad l \leftarrow DetermineJumpMagnitude(l, s, history)$
6: $\quad t \leftarrow DeterminePerturbationType(s, \mathbf{e}, history)$
7: $\quad s' \leftarrow Perturbation(l, t, s, history)$
8: **end while**

random, the probability vector $\mathbf{e}$ is a set of all the probabilities e of selecting a directed over a random perturbation type. Once the number l and the type t of perturbation moves are selected, BLS calls the *Perturbation* procedure which applies l moves of type t to the current local optimum (we say that the solution is perturbed). The perturbed solution becomes the new starting point for the next phase of the descent-based local search. The *history* element in *DetermineJumpMagnitude*, *DeterminePerturbationType* and *Perturbation* indicates that some information from the search history is used to influence the decisions made in these procedures.

In all the previous applications of BLS, *DetermineJumpMagnitude* and *DeterminePerturbationType* procedures are mutually exclusive, and the values for l and e are thus determined independently from each other. As a result, the combination of l and e may not constitute the optimal degree of diversification required at one stage of the search.

3. Breakout Local Search based on Reinforcement Learning (BLS-RLE)

The main novelty of BLS-RLE lies in a dedicated perturbation control mechanism that uses ideas from reinforcement learning [27] to interdependently determine the values for the number of perturbation moves l and the probability e of selecting one perturbation type over another. More precisely, the proposed mechanism is based on an adaptive operator selection paradigm [37, 38, 39], where the application of each action is followed by an immediate reward from the environment (the problem). This instantaneous reward is used for calculating an action value which provides a quality estimate of the corresponding action based on its historical performance. An action selection model is then employed to select the next action to be applied. Given the space limitation, we next describe only the new perturbation strategy of BLS-RLE. The interested reader is referred to [22] for a

detailed description of the other algorithmic components of BLS-RLE (i.e., *DescentBasedSearch*, *GenerateInitialSolution*, hash table structure, directed and random perturbation moves), which are identical to those used by BLS for VSP.

3.1. General framework of BLS-RLE

Given a finite set of values $L = \{l_0, ..., l_{h-1}\}$ for the number of perturbation moves and a finite set of values $E = \{e_0, ..., e_{k-1}\}$ for the probability of selecting a directed over a random perturbation type, let $P_{all} = \{(l, e)_0, ..., (l, e)_{h \cdot k-1}\}$ be the set of all the possible combinations (parameter pair settings) of L and E. We use the terms *action* and *parameter pair* interchangeably throughout the paper to denote $(l, e) \in P_{all}$.

The proposed RL-based parameter control technique favors actions that introduce a weaker diversification when the search is located in an unvisited region of the search space. As the search keeps going back and forth between already visited local optima, the preference shifts towards actions that result in a higher degree of diversification with the aim to discover new local optima. Therefore, the pairs in P_{all} need to be sorted according to the degree of diversification they introduce into the search, prior to the optimization phase. This is performed by a *PrelearningPhase* procedure.

To limit the number of considered actions and to reduce the learning time, our technique uses an action selection model that selects the next action (l, e) for perturbation only from a very limited learning set $P_{learn}\{(l, e)_0, ..., (l, e)_{\kappa-1}\} \subset P_{all}$ of possible parameter pair configurations, where κ is a constant. The set of allowed actions is periodically being updated with a new and more preferred action for a given stage of the search. According to an analysis provided in Section 5.2.3, sampling of the allowed action space constitutes one of the key features of the proposed parameter controller. The proposed BLS-RLE framework is presented in Algorithm 2.

BLS-RLE first calls the *PrelearningPhase* procedure (line 3 of Algorithm 2, see Section 3.2) to assign a rank to each pair $(l, e) \in P_{all}$, according to the estimated diversification strength that each pair introduces into the search. To evaluate the amount of introduced diversification for each $(l, e) \in P_{all}$, BLS-RLE maintains a set of recently visited local optima in a hash table *HT* memory (lines 9 and 17 of Algorithm 2, see [22]). The *PrelearningPhase* procedure is called only once, before the main run of the proposed algorithm. After pre-learning, a limited subset of allowed actions $P_{learn} \subset P_{all}$ is initialized with κ pairs that significantly differ in the amount of diversification they introduce into the search (line 4 of Algorithm 2). To generate a starting

Algorithm 2 Improved BLS framework (BLS-RLE)

1: Let $L = \{l_0, ..., l_{h-1}\}$ be the set of possible values for the number of perturbation moves
2: Let $E = \{e_0, ..., e_{k-1}\}$ be the set of possible values for the probability of applying directed over random perturb.)
3: $P_{all} \leftarrow PrelearningPhase(L, E)$ /* See Section 3.2*/
4: $P_{learn} \leftarrow \{(l, e)_0, ..., (l, e)_{\kappa-1}\}$ /* Initial set of κ learning parameter pairs sorted in ascending order according to the degree of diversification introduced */
5: $iter \leftarrow 0$
6: $HT \leftarrow \emptyset$
7: $s' \leftarrow GenerateInitialSolution$
8: $s \leftarrow DescentBasedSearch(s')$
9: $HT \leftarrow UpdateHashTable(s)$
10: **while** stopping condition not reached **do**
11: $i \leftarrow SelectParameterPair()$ /*See Section 3.4)*/
12: $(l, e) \leftarrow (l, e)_i \in P_{learn}$
13: $s' \leftarrow Perturbation(l, e, s, history)$
14: $s \leftarrow DescentBasedSearch(s')$
15: $r \leftarrow DetermineParameterPairReward(s, HT)$ /*See Section 3.3*/
16: $ApplyReward(i, r)$ /*See Section 3.3*/
17: $HT \leftarrow UpdateHashTable(s)$
18: **if** $(iter > \epsilon)$ **then**
19: $UpdateLearningParameterSet()$ /*See Section 3.5*/
20: $iter \leftarrow 0$
21: **else**
22: $iter \leftarrow iter + 1$
23: **end if**
24: **end while**

point for the search, BLS-RLE calls *GenerateInitialSolution* to obtain a solution s', which is then improved with a descent-based local search procedure to obtain a local optimum s (lines 7-8 of Algorithm 2, see [22]).

After initialization, each iteration of BLS-RLE performs the following steps. First, BLS-RLE employs the action selection model through the *SelectParameterPair* procedure (lines 11-12 of Algorithm 2, see Section 3.4), to select a parameter pair $(l, e)_i \in P_{learn}$. The pair $(l, e)_i \in P_{learn}$ is selected proportionally to its action probability, which is based on the action value of the corresponding parameter pair configuration. We employ the Softmax action-selection rule [27] as the action selection model. However, notice that other action selection models can be easily deployed instead. The *Perturbation* procedure is then called with the selected $(l, e)_i$ to perturb the current solution (line 13 of Algorithm 2). This is immediately followed by the descent-based local search which repairs the perturbed solution and reaches a local optimum s (line 14 of Algorithm 2). Next, BLS-RLE determines

the success (immediate reward) of applying the selected pair $(l, e)_i$ with respect to s (line 15 of Algorithm 2, see Section 3.3), and updates the action value attributed to the pair $(l, e)_i$ with the acquired reward (line 16, see Section 3.3). The action value of a parameter pair configuration $(l, e)_i$ takes into account the recent immediate rewards awarded to $(l, e)_i$ that are determined based on a quality and a diversity criterion. Initially, action values for all $(l, e) \in P_{learn}$ are set to 1. Finally, the *UpdateLearningParameterSet* procedure is periodically called (after ϵ iterations of BLS-RLE) to update P_{learn} with a new parameter pair from P_{all} (line 18-23 of Algorithm 2, see Section 3.5), considering the learned action probabilities corresponding to the current parameter pairs in P_{learn} and the parameter pair ranks determined in the pre-learning phase. After a call of *UpdateLearningParameterSet*, all the action values are reset to 1.

3.2. Parameter Pair Pre-learning

Algorithm 3 *PreLearning*(L, E)

Require: Set $L = \{l_0, ..., l_{h-1}\}$ of values for the number of perturbation moves and set $E = \{e_0, ..., e_{k-1}\}$ of values for the probability of applying a directed over a random perturbation

Ensure: Parameter pair list $P_{all} = \{(l, e)_0, ..., (l, e)_{h \cdot k-1}\}$ sorted according to the degree of diversification introduced by each pair

1: $P_{all} \leftarrow$ *GenerateCombinations*(L, E) /* Creates the set of all $h \cdot k$ parameter pairs */
2: $c[0, ..., h \cdot k - 1] \leftarrow 0$ /* For each parameter pair, initialize the number of encounters of an already visited solution from hash memory */
3: $s' \leftarrow$ *GenerateInitialSolution*
4: $s \leftarrow$ *DescentBasedSearch*(s')
5: **for** j:=0 to $\alpha \cdot h \cdot k - 1$ **do**
6: $(l, e) \leftarrow (l, e)_{j \bmod (h \cdot k)} \in P_{all}$
7: $s' \leftarrow$ *Perturbation*$(l, e, s, history)$
8: $s \leftarrow$ *DescentBasedSearch*(s')
9: **if** $(HT(s))$ /* Is s in hash table memory? */ **then**
10: $c[j \bmod (h \cdot k)] \leftarrow c[j \bmod (h \cdot k)] + 1$
11: **end if**
12: **end for**
13: Sort P_{all} in ascending order according to c values

The purpose of pre-learning is to estimate the degree of diversification introduced with each parameter pair $(l, e) \in P_{all}$, where the degree of diversification is measured by the frequency the search encounters new solutions after an application of (l, e). Therefore, for each available parameter pair, a standard iterated local search algorithm [24] is employed to capture the

degree of diversification of the pair under consideration, as demonstrated in Algorithm 3. Specifically, starting from an initial locally optimal solution, it alternates between a perturbation phase to perturb the current local optimum and a descent-based local search phase (*DescentBasedSearch* is detailed in [22]) to reach a local optimum. Given a parameter pair setting (l, e), the perturbation phase simply consists of performing l moves of the directed (tabu-based) perturbation [22] with probability e, and l moves of the random perturbation [22] otherwise. The ILS procedure terminates after $\alpha \cdot h \cdot k$ iterations, where α is a coefficient which denotes the number of times each parameter pair setting (l, e) is used for the perturbation phase (in our experiments $\alpha = 100$). Let $(l, e)_{j \bmod (h \cdot k)}$ be the parameter pair setting used for perturbation at iteration j, the procedure increments the counter $c[j \bmod (h \cdot k)]$ corresponding to $(l, e)_{j \bmod (h \cdot k)}$ if the descent-based search returned to an already encountered local optimum stored in the hash table memory.

Note that the best solution found in the pre-learning phase could serve as the starting point for the main BLS-RLE search (i.e., instead of the local optimum generated in lines 7-8 of Algorithm 2).

In the end, parameter pairs in P_{all} are sorted in an ascending order according to the learned degree of diversification. The rank of a parameter pair configuration $(l, e)_i \in \{(l, e)_0, ..., (l, e)_{h \cdot k-1}\}$ corresponds to its order in P_{all}. These ranks are later required for sampling the action space, as well as for the reward function.

Finally, it is worth mentioning that the proposed RL-based parameter controller is able to ensure a meaningful guidance for the search if and only if the structure of the problem landscape is such that different (l, e) pairs introduce different degrees of diversification into the search (e.g., in case of rugged landscapes). In cases of landscapes with large plateau regions, the controller loses its power and could result in a performance worse than that of a random parameter selection mechanism.

3.3. Reward and Value Functions

After a perturbation with action $(l, e)_i \in P_{learn}$ followed immediately by a descent-based search to reach a local optimum s, the quality or value of the employed action has to be estimated. The action value is updated based on the reward r_i that captures the immediate impact of $(l, e)_i$ on the search process. Intuitively, the action value indicates the total amount of reward an agent can expect to accumulate over the future.

To determine the reward r_i, we take into account both the *quality* and the *diversity* criteria. The higher the score assigned to $(l, e)_i$, the better

the performance of $(l, e)_i$ in the last perturbation phase. Notice that the parameter pairs in $P_{learn} \subset P_{all}$ are ordered in an increasing order of their rank, determined in the pre-learning phase (see previous Section).

The element of the reward for *diversity* r_{div} can be obtained with the following formula:

$$r_{div} = \begin{Bmatrix} 0, & \text{if } HT(s) \\ \kappa - i, & \text{otherwise} \end{Bmatrix}, \tag{1}$$

where $HT(s)$ returns the value *true* if s is present in the hash table HT. The rationale behind Eq. 1 is to introduce the least amount of diversification sufficient to discover new local optima. Therefore, the smaller the index i of the parameter setting pair $(l, e)_i$ (i.e., the closer i is to zero), the higher the score of the reward. In this way, we prevent from using a parameter pair setting $(l, e) \in P_{learn}$ which unnecessarily introduces too much diversity into the search. The reward function thus complies with the principle "intensification first, minimal diversification only if needed" [35], and constitutes one of the key features of the proposed parameter controller. More precisely, the proposed perturbation mechanism is biased towards lower ranked parameter pair settings (pairs that introduce a lower degree of diversification) as the search keeps discovering new local optima. On the other hand, the rank of the selected parameter pairs (i.e., the degree of diversification) increases with the increase of repetitions of already encountered local optima.

The reward for *quality* r_{qual} is determined according to the next equation:

$$r_{qual} = \begin{Bmatrix} 0, & \text{if } HT(s) \\ (1 - \frac{f(s) - f(s_{best})}{f(s_{best})})^2 \cdot 10, & \text{otherwise} \end{Bmatrix}, \tag{2}$$

where $f(s)$ and $f(s_{best})$ are respectively the objective function values of the current and the best solution found during the search until the current iteration. The quality reward takes into account the relative fitness gain of the current solution s with respect to the best solution found s_{best}, and distributes it in a quadratic manner, i.e., $10(1-x)^2$, where x is the relative fitness gain. The rationale is to favor parameter pair settings which lead to undiscovered high quality local optima with lower deviations from the best solution found.

Given rewards r_{div} and r_{qual} for the parameter pair setting $(l, e)_i \in P_{learn}$, the total reward r_i attributed to $(l, e)_i$ is

$$r_i = \delta_1 \cdot r_{div} + \delta_2 \cdot r_{qual}, \tag{3}$$

where δ_1 and δ_2 are two coefficients that determine respectively the impact of the diversity r_{div} and the quality r_{qual} of the overall reward r_i.

Once the reward r_i for the performance of $(l, e)_i$ in the last perturbation is computed, BLS-RLE calls the *ApplyReward*(i, r_i) function which updates the credit (i.e., the empirical quality estimate $\hat{r}_i$) attributed to $(l, e)_i$ with its latest reward r_i. The empirical quality estimate can be defined as the performance summary of an action, and is used to compute the action value q corresponding to the given parameter pair configuration. One could employ various different ways to update the empirical quality estimate [37], including instantaneous, average, or rank based rewards. The simplest is to consider the most recent reward r_i as the empirical quality estimate $\hat{r}_i$ for $(l, e)_i$. However, this tends to be a very unstable and noisy estimation [37].

In our work, the empirical quality estimate $\hat{r}_i$ assigned to a given parameter pair setting $(l, e)_i \in P_{learn}$ is the average of the w latest rewards attributed to $(l, e)_i$ [37], where $w = 100$ in our experiments. This results in an aggregation of $0 \leq w \leq 100$ latest rewards, which constitutes a more stable and noise-tolerant credit for $\hat{r}_i$. More precisely, let R_i be a set of the least recent rewards for $(l, e)_i$. The empirical quality estimate for the $(l, e)_i$ pair is calculated as: $\hat{r}_i = \sum_{j=1}^{|R_i|} R_i(j)/|R_i|$, where $|R_i|$ denotes the cardinality of the set R_i, and $R_i(j)$ is the j-th element of the R_i set.

Finally, we estimate the action value $q_{a_i}(t+1)$, corresponding to action $(l, e)_i$ (or a_i for short), with the following update rule:

$$q_{a_i}(t+1) = q_{a_i}(t) + \lambda(\hat{r}_i(t) - q_{a_i}(t)),$$

where t is the time step and $\lambda \in (0, 1]$ is the adaptation rate (λ is fixed to 0.1). For $t = 0$, the value of $q_{a_i}(t)$ is initialized to 1.

3.4. Parameter Pair Selection with a Reinforcement Learning Approach

To select an action for the next perturbation phase, BLS-RLE utilizes the Softmax action-selection approach [27] (*SelectParameterPair* procedure at line 11 of Algorithm 2) which uses a trial-and-error learning methodology to actively explore the available actions.

More precisely, let $\mathcal{A} = \{a_1, a_2, \ldots, a_\kappa\}$ be the set of κ possible actions (i.e., parameter pair configurations) and let $Q(t) = \{q_{a_1}(t), q_{a_2}(t), \ldots, q_{a_\kappa}(t)\}$ be the set of action values corresponding to the action set A at the time step t. The Softmax-based approach assigns a probability to each action a_i by using the Gibbs (or Boltzmann) distribution for the assignment. More precisely, we define the action probability $p_{a_i}(t)$ according to the following

equation:

$$p_{a_i}(t) = \frac{e^{q_{a_i}(t)/\tau}}{\sum_{j=1}^{\kappa} e^{q_{a_j}(t)/\tau}},$$

where τ is a positive parameter called *temperature*. High values for τ result (almost) equal probabilities for all the available actions, while low temperature values induce larger differences in the selection probabilities. Therefore, the best performing actions are selected with higher probabilities in the next step. It is useful to mention that, as the temperature decreases (approaches to zero), the Softmax action selection rule becomes very greedy and tends to always select the best performing actions.

Having calculated the probabilities of all the available choices, we select the pair $(l, e)_i \in P_{learn}$ according to a stochastic selection procedure, in the form of a simple roulette wheel selection procedure [40].

The main steps of the Softmax methodology for selecting parameter setting pairs are general and follow the concept of adaptive (or dynamic) operator selection [37]. Thus, several other methodologies can be employed instead of Softmax, including evolutionary meta-learning methods [41, 42], probabilistic models such as probability matching [39], adaptive pursuit [39], statistical based models like the multinomial distribution tracking with history forgetting [38, 43], and reinforcement learning approaches [37, 27]. However, the application and comparison of all these approaches is out of the scope of this article and will be studied in a future work.

3.5. Update of the Parameter Pair Learning List

After a fixed number of BLS-RLE iterations, we replace the worst parameter pair setting $(l, e)_w \in P_{learn}$ (i.e., the one with the lowest action probability) with a new and potentially better pair $(l, e)_n \in P_{all}$, $(l, e)_n \notin P_{learn}$. Recall that $P_{learn} \subset P_{all}$ is maintained to reduce the action space (i.e., the number of available parameter pair choices) resulting in shortened learning time. In fact, one of the keys elements to success for the proposed parameter controller is the strategy for action space sampling, which enables rapid convergence towards a limited set of actions (P_{learn}) that appear to be the most convenient for the given state of search (see Section 5.2.3).

To determine $(l, e)_n$ for P_{learn}, we first estimate the action probabilities for all the parameter pair settings from $P_{all} \setminus P_{learn}$, based on the previously learned action probabilities attributed to the pairs from P_{learn}. For this purpose, we simply take the action probabilities of all the pairs in P_{learn}, and interpolate them linearly to sample the action probabilities corresponding to pairs $P_{all} \setminus P_{learn}$. The boundaries for the linear interpolation are set as

follows: for $(l,e)_0 \in P_{all}$ the estimated action probability is computed as $p_{(l,e)_0} = p_f/i$, where p_f is the action probability corresponding to $(l,e)_0 \in P_{learn}$ and i is the rank (order number) of $(l,e)_0 \in P_{learn}$ in P_{all}; for $(l,e)_m \in P_{all}$ the estimated action probability is computed as $p_{(l,e)_m} = p_l/(m-j)$, where p_l is the action probability corresponding to the last pair $(l,e)_{\kappa-1} \in P_{learn}$, $m = |P_{all}| - 1$, and j is the rank (order number) of $(l,e)_{\kappa-1} \in P_{learn}$ in P_{all}. An example of this procedure is depicted in Figure 1.

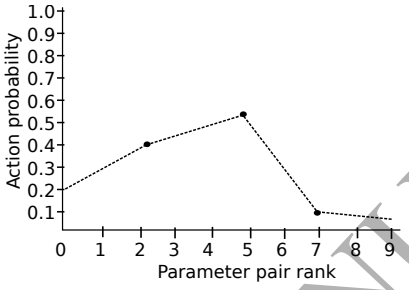


Figure 1: Linearization of action probabilities, corresponding to the parameter setting pairs in P_{learn} ($\kappa = 3$), to sample action probabilities for pairs $P_{all} \setminus P_{learn}$ (the number of all the possible parameter setting pairs in P_{all} is 10).

Next, we sort the parameter pairs from $P_{all} \setminus P_{learn}$ in a decreasing order according to their estimated action probability. A parameter pair setting $(l,e)_n \in \{P_{all} \setminus P_{learn}\}$ is then selected in an adaptive and random way, i.e., the higher the estimated action probability of a parameter pair setting, the more possibly this pair is chosen to replace $(l,e)_w \in P_{learn}$. More precisely, the i^{th} best parameter pair setting from $P_{all} \setminus P_{learn}$ is selected according to the following probability function:

$$N(i) = i^{\phi} / \sum_{j=1}^{k} j^{\phi}, \tag{4}$$

where k is the total number of pairs in $P_{all} \setminus P_{learn}$, and ϕ is a negative real number which controls the intensity of the selection procedure. The smaller the value of ϕ, the higher the possibility to select a parameter pair setting with a higher action probability.

Once P_{learn} is updated with $(l,e)_n$, BLS-RLE sorts P_{learn} in an ascending order according to the amount of diversification introduced by each pair $(l,e) \in P_{learn}$. Finally, the action values for all $(l,e) \in P_{learn}$ are reset to 1.

4. Experimental Results and Comparisons

In this section, we demonstrate the performance of the proposed approach on a wide variety of problem instances with different characteristics. Before proceeding with experimental evaluations, we first define the experimental protocol and present the benchmark sets used for comparisons.

4.1. Experimental Protocol

The main objective of this section is to analyze the performance of the proposed RL-based perturbation mechanism compared to the existing BLS perturbation mechanism, and the standard perturbation strategies commonly used by the ILS algorithms. More precisely, we wish to evaluate the ability of the method to self-adaptively establish the most suitable balance between intensification and diversification, for an instance or a benchmark set at hand, without any particular need for human intervention. The ability to self-adapt its performance for a given instance at hand constitutes the essence of reactive search methods [35]. We carry out comparisons between BLS-RLE and the following algorithms:

- BLS - the existing breakout local search proposed in [22];
- BLS-RND - a BLS algorithm which, for each perturbation phase, randomly selects a value for e from the range $[0.95, 1]$ and a value for l from the range $[3, 150]$. Random variation of the search parameters is adopted as "base-line" [44] to evaluate the impact and the usefulness of the proposed self-adaptive mechanism.
- ILS-DIRP - an iterated local search algorithm which combines a simple descent procedure with the directed perturbation (l is fixed to 40).
- ILS-RNDP - an iterated local search algorithm which combines a simple descent procedure with the random perturbation (l is fixed to 40).
- VNS - a recent variable neighborhood search algorithm from [23]. However, results obtained with VNS are available only for a subset of instances used in the following experiments.

All algorithms[1] are implemented in C++ and compiled with GNU g++ under GNU/Linux running on a machine equipped with AMD Opteron CPU

[1]The source code of BLS-RLE is available at: `http://www.epitropakis.co.uk/BLS-RLE/`

of 2.2 GHz and 32 GB of RAM. The results reported by BLS-RLE and the first four algorithms are obtained under the same computing conditions. It is worth noting that BLS-RND, ILS-DIRP and ILS-RNDP can be viewed as simplified versions of the existing BLS algorithm (i.e., variations of ILS), and were implemented solely for the purpose of the reported experimental comparisons. The experiments for VNS are taken from the corresponding paper, and are only used for indicative purposes. The main emphasis is thus put on the comparison with BLS, BLS-RND, ILS-DIRP and ILS-RNDP.

To demonstrate the self-adaptive capabilities of BLS-RLE, we do not fine-tune the parameters of BLS-RLE. The setting of parameters for BLS-RLE, determined mostly based on our intuition and some manual trials, is given in Table 1. A parameter sensitivity analysis (available online[2]) indicates that the most sensitive parameters are the temperature (τ) of the Softmax-based adaptive procedure, as well as the diversity (δ_1) and quality (δ_2) coefficients of the reward function. Temperature values close to $\tau = 2$ seem to lead to very good performance gains, while very stable behavior is exhibited when $\delta_1 \in \{0.5, 1, \ldots, 3\}$ and $\delta_2 \in \{2, 2.5, 3\}$. This indicates that the contribution of the quality reward component should be amplified as it provides valuable guidance for the BLS-RLE search. The remaining 6 parameters (α, ϵ, κ, ϕ, w, and λ) do not exhibit a statistically significant impact on the resulting performance of BLS-RLE. However, fine-tuning of some of these parameters may result in a slight improvement of the algorithm performance. In our experiments with BLS-RLE, the set P_{all} is populated with all the combinations of $L = \{3, 5, \ldots, i-2, i, i+2, \cdots, 147, 149\}$ and $E = \{0.95, 0.96, 0.97, 0.98, 0.99, 1\}$. Note that the ranges of values from which BLS-RND selects a value for l and e are the same as in the case of BLS-RLE. For BLS, we use the configuration of parameters suggested in the corresponding paper [22], while the tuning procedure to determine the l parameter for ILS-DIR and ILS-RND is the same as the one used for tuning BLS in [22]. For each algorithm and each problem instance, we conduct 30 independent runs, with the time limit set to 2000 seconds (and 1800 seconds for instances used in the comparison with VNS as in [23]). The pre-learning phase is executed only once, and is not accounted for in each of the 30 independent runs. The time overhead incurred by the pre-learning phase, as well as the possibility of a warm-start pre-learning procedure, are investigated in Section 5.2.2.

[2]http://www.epitropakis.ac.uk/BLS-RLE

Table 1: Parameter settings of BLS-RLE.

Parameter	Description	Value
κ	the size of the learning parameter pair set P_{learn}	6
α	the number of pre-learning iterations for each pair in P_{all} (Section 3.2)	100
τ	the temperature of the Softmax-based adaptive procedure (Section 3.4)	2
ϵ	the update frequency of the parameter pair learning list (Section 3.5)	2000
δ_1	coefficient for reward function (Section 3.3)	1.5
δ_2	coefficient for reward function (Section 3.3)	3
γ	tabu tenure of DIRP	$random(0.2\|C\|, 0.7\|C\|)$
ϕ	coefficient used by the strategy that updates P_{learn} (Section 3.5)	2
w	the number of latest rewards considered for computing the empirical quality estimate (Section 3.3)	100

4.2. Benchmark instances

The existing VSP benchmark set [13] consists of 104 instances with $11 \leq |\mathcal{V}| \leq 191$, $20 \leq |\mathcal{E}| \leq 13922$, the size constraint $b = 2|\mathcal{V}|/3$, and with densities in the range of $[0.05, 0.93]$. A total of 84 instances were adapted from the well-known Market Matrix repository[3], which consists of intersection graphs obtained from the coefficient matrices of linear equations, while 20 instances come from the DIMACS challenge on graph coloring. Optimal solutions are known for the complete benchmark set. A detailed description and motivation for these instances can be found in [13]. However, it has been shown in [22, 23] that this benchmark set does not represent a real challenge for heuristic approaches, since BLS and VNS can often solve these instances to optimality in less than a second. To evaluate the performance of BLS-RLE with respect to the reference algorithms, we thus introduce a large set of challenging instances (426 instances in total) with different properties (i.e., sizes, densities, applications) from the following benchmark sets:

B1: The ISPD98 Circuit Benchmark Suite[4] arising from VLSI design (one of the first applications of VSP [8, 2, 3]). This benchmark set consists of 17 very large graphs with $12752 \leq |\mathcal{V}| \leq 185495$, $14111 \leq |\mathcal{E}| \leq 189581$, $b = 2|\mathcal{V}|/3$ and a maximal density of 0.0782.

B2: Scale-free network graphs generated with the Barabási-Albert model

[3] `http://math.nist.gov/MatrixMarket/`

[4] `http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html`

[45, 46]: Scale-free networks are widely observed in natural and human-made systems, including the internet, the world wide web, citation and social networks. The choice of this benchmark set is motivated by the application of VSP to cyber security and decoy routing [5, 6]. It comprises 95 instances from [23], with $100 \leq |\mathcal{V}| \leq 1000$, $b = 2|\mathcal{V}|/3$, and the node degree d randomly selected from $[1, \mathcal{V}]$. These instances were used to evaluate the performance of a VNS algorithm in the corresponding paper. We further generate an additional set of 85 more challenging Barabási-Albert instances with $1000 \leq |\mathcal{V}| \leq 2000$, $b = 2|\mathcal{V}|/3$, $d \in [3, 100]$, and density in the range of $[0.03, 0.897]$.

B3: Exponential network graphs generated by the Erdös-Rényi model [47]: Unlike scale-free networks, exponential networks are homogeneous, i.e., most nodes have approximately the same number of links. Each graph is denoted as $G(\mathcal{V}, p)$, where $\mathcal{V}$ is the number of vertices and p is the probability of connecting a pair of vertices. Obviously, the density of a graph increases with the value of p. Our benchmark set consists of 95 instances taken from [23], with $100 \leq |\mathcal{V}| \leq 1000$, $b = 2|\mathcal{V}|/3$ and $p \in [0.2, 0.1]$. Moreover, we introduce 45 more challenging Erdös-Rényi instances with $1000 \leq |\mathcal{V}| \leq 5000$, $p \in [0.01, 0.5]$, and density in the range of $[0.0198, 0.977]$.

B4: A set of graphs taken from the Stanford SNAP database[5] (16 instances) and the University of Florida Sparse Matrix Collection[6] (19 instances): The selected instances from the SNAP database include internet peer-to-peer networks, as well as instances arising from communication networks, web graphs, and social networks. These graphs are very large with $3809 \leq |\mathcal{V}| \leq 122749$. The selected instances from the Sparse Matrix Collection were derived from different real applications including co-authorship and protein interaction networks, with $1000 \leq |\mathcal{V}| \leq 11125$, $b = 2|\mathcal{V}|/3$ (as in $B1$), and density in the range of $[0.000131, 0.0132]$.

B5: A set of 54 benchmark instances[7] consisting of toroidal, planar, and random graph with $800 \leq |\mathcal{V}| \leq 3000$, and density ranging from 0.00133 to 0.06. These graphs were generated in [48] to test methods for solving semidefinite programming problems, and adapted in [22] to

[5] `http://snap.stanford.edu/data/`

[6] `http://www.cise.ufl.edu/research/sparse/matrices/`

[7] Instances available at: `http://www.info.univangers.fr/pub/hao/BLSVSP.html`

evaluate BLS on the VSP problem.

All benchmark sets are publicly available at [8], while the hardness of the benchmark sets is analyzed in Section 5.

4.2.1. Experimental and Statistical Results

This section summarizes the performances obtained with the implemented algorithms on the five benchmark sets (B_1 to B_5) from Section 4.2, and statistically verifies the observed behavior. Due to space limitations and the large number of considered benchmarks, a more detailed presentation of individual results per problem instance is available at: `http://www.epitropakis.co.uk/BLS-RLE/`.

To compare the performances (in terms of solution quality, i.e., attained objective function values) of all the algorithms across all the problem instances from a given benchmark set and across every execution, we normalize the obtained objective values in a common range of values. Given a problem instance, this is achieved with a linear normalization procedure that takes an objective value defined on a range $D = [O_{\min}, O_{\max}]$ and transforms it linearly to a new range $F = [o_{\min}, o_{\max}]$ (here $F = [0, 1]$), where $O_{\min}$ and $O_{\max}$ is the minimal and maximal observed objective value respectively. Formally, let $x \in [A, B]$ be an objective value returned by an algorithm for an instance at hand and let $f : D \to F$ be a transformation function, the normalized objective value can be calculated according to the following relation: $y = f(x) = (x - O_{\min})/(O_{\max} - O_{\min})$. For each algorithm Alg, we then compute a sample set $Y_{Alg} = \{y_1, y_2, \ldots, y_n\}$ consisting of the normalized objective values across all the benchmark instances and runs, where n is the number of problem instances from a benchmark set times the number of executions of the given algorithm per instance. Intuitively, values close to zero indicate the best performance, while the performance is worse as the values increase to one. We provide boxplot graphs to compare the distributions of the normalized objective values in Y_{Alg} obtained with each algorithm.

To assess whether there exists a significant difference in the observed performances between at least two algorithms on the complete set of benchmark instances, we first employ the Friedman rank sum test [49] on all sample sets $M = \{m_1, m_2, \ldots, m_b\}$, where M represents the set of normalized mean performance values obtained with a given algorithm for each instance from a

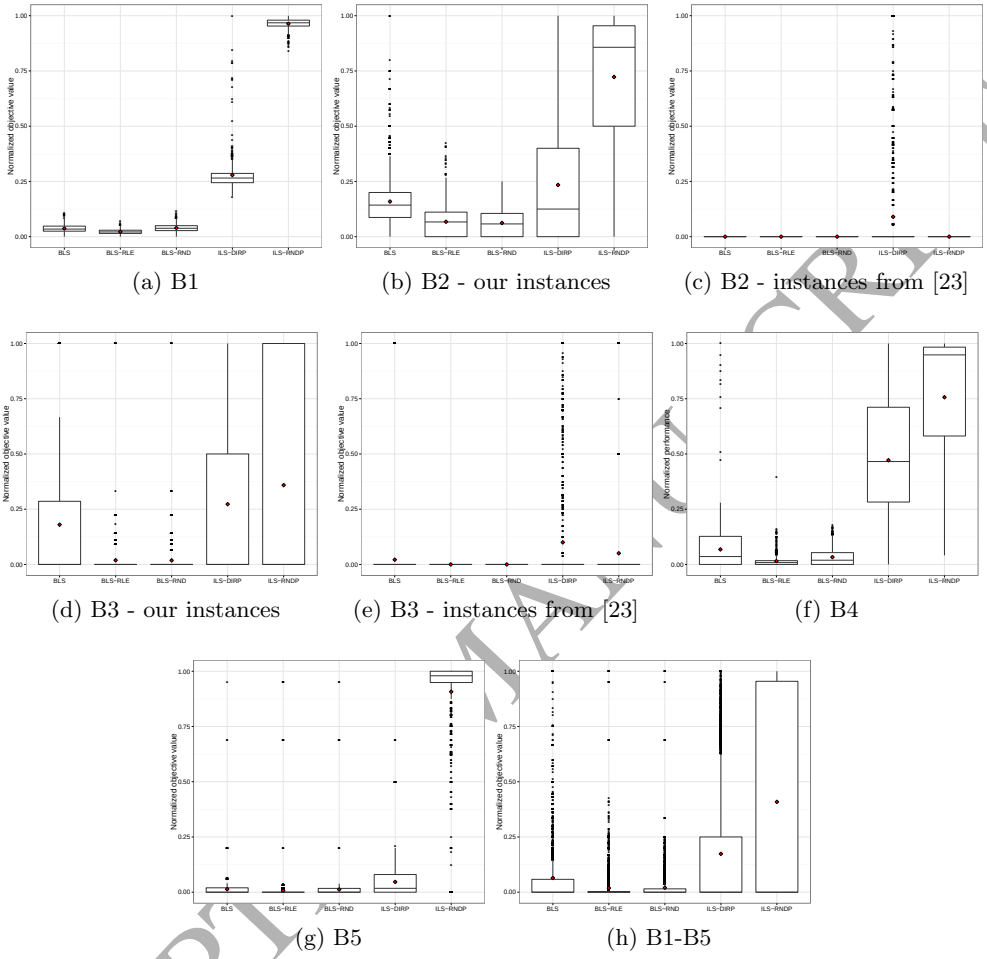
[8] `http://www.epitropakis.co.uk/BLS-RLE/`

given set of b benchmark instances. The null hypothesis of the Friedman rank sum test states that the underlying distributions of all samples M are the same, while the alternative hypothesis states that at least two samples are not the same [49]. If significant difference in performances exists, we carry out a post-hoc analysis to determine which two algorithms differ in performance. For the post-hoc analysis, we perform pairwise Wilcoxon-signed rank tests on the sample sets M. Additionally, to alleviate from having Type I errors in multiple comparisons with a higher probability, we apply the Bonferroni correction method and report the adjusted p-values (p_{bonf}).

	BLS		BLS-RLE		BLS-RND		ILS-DIRP	
	p	p_{bonf}	p	p_{bonf}	p	p_{bonf}	p	p_{bonf}
Benchmark B1 (Friedman: $p < 0.0001$, $\chi^2 = 61.741$)								
BLS-RLE	**0.002**	**0.002**	–	–	–	–	–	–
BLS-RND	0.013	1.000	**0.001**	**0.010**	–	–	–	–
ILS-DIRP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
Benchmark B2 (Friedman: $p < 0.0001$, $\chi^2 = 230.731$)								
BLS-RLE	**<0.001**	**<0.001**	–	–	–	–	–	–
BLS-RND	**<0.001**	**<0.001**	0.258	1.000	–	–	–	–
ILS-DIRP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
Benchmark B3 (Friedman: $p < 0.0001$, $\chi^2 = 134.479$)								
BLS-RLE	**<0.001**	**<0.001**	–	–	–	–	–	–
BLS-RND	**<0.001**	**<0.001**	0.888	1.000	–	–	–	–
ILS-DIRP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**0.007**	0.073
Benchmark B4 (Friedman: $p < 0.0001$, $\chi^2 = 115.343$)								
BLS-RLE	**<0.001**	**0.002**	–	–	–	–	–	–
BLS-RND	**<0.001**	**0.005**	**0.001**	**0.014**	–	–	–	–
ILS-DIRP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**0.004**
Benchmark B5 (Friedman: $p < 0.0001$, $\chi^2 = 154.347$)								
BLS-RLE	**<0.001**	**<0.001**	–	–	–	–	–	–
BLS-RND	0.453	1.000	**<0.001**	**<0.001**	–	–	–	–
ILS-DIRP	**<0.001**	**0.002**	**<0.001**	**<0.001**	**<0.001**	**0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
Benchmark B1-B5 (Friedman: $p < 0.0001$, $\chi^2 = 610.000$)								
BLS-RLE	**<0.001**	**<0.001**	–	–	–	–	–	–
BLS-RND	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–	–	–
ILS-DIRP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
ILS-RNDP	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**

Table 2: Post-hoc analysis of the mean normalized performances reported with BLS-RLE and the reference algorithms across all the individual benchmark sets $B1$-$B5$ and the combined benchmarks.

For each benchmark set $B1$-$B5$, Figure 2 provides boxplot graphs (a)-(g) to show the distribution of the normalized performances ($Y_{BLS-RLE}$, Y_{BLS},



(a) B1 (b) B2 - our instances (c) B2 - instances from [23]

(d) B3 - our instances (e) B3 - instances from [23] (f) B4

(g) B5 (h) B1-B5

Figure 2: Boxplots of the normalized objective values for each individual benchmark set $B1$-$B5$ ((a)-(g)), as well as for the combination of all benchmark sets (h).

$Y_{BLS-RND}$ $Y_{ILS-DIRP}$, $Y_{ILS-RNDP}$) corresponding to the five algorithms. For the sake of clarity, we include two boxplots per benchmark for $B2$ ((b) and (c)) and $B3$ ((d) and (e)) to separate the presentation of our instances from those taken from [23]. Moreover, subfigure (h) provides a boxplot to summarize the overall performances across all the instances from $B1$-$B5$. The Friedman rank sum test reveals a statistically significant difference in performances between the considered algorithms across all the benchmark sets $B1$-$B5$. To determine which two algorithms differ in performance, we

thus continue with the post-hoc analysis and show in Table 2 the obtained p-values from the Wilcoxon-signed rank tests (column 'p') and the Bonferroni correction method (column 'p_{bonf}'). Moreover, for each comparison, we include in Table 2 the p and χ^2 values reported by the Friedman rank sum test. From boxplots (a), (f) and (g), we observe that BLS-RLE outperforms the four reference algorithms on the benchmark sets $B1$, $B4$ and $B5$. Indeed, we observe that the normalized objective values obtained with BLS-RLE are very close to zero. The p-values from Table 2 also confirm a significant statistical difference in performance between BLS-RLE and the reference methods on these benchmarks. For the other two benchmark sets ($B2$ and $B3$), the boxplots (b)-(e) and the p-values indicate an almost equal performance of BLS-RLE and BLS-RND, which however remains superior to that reported by BLS, ILS-DIRP and ILS-RNDP. Surprisingly, BLS-RND shows to be more effective than BLS across all the benchmark sets, which emphasizes the weakness of the current BLS perturbation mechanism. These observations support the study from [44] that suggests considering random variation of parameters when evaluating a new parameter control mechanism.

	B2: Barabási-Albert				B3: Erdös-Rényi			
Algorithm	Avg.	Time(s)	% Dev.	# Best	Avg.	Time(s)	% Dev.	# Best
BLS	**278.063**	0.790	0.000	95	291.031	0.576	0.010	95
BLS-RLE	**278.063**	3.164	0.000	95	**291.010**	1.722	0.000	95
BLS-RND	**278.063**	3.234	0.000	95	**291.010**	1.920	0.000	95
ILS-DIRP	278.907	105.853	0.002	95	291.943	37.991	0.242	93
ILS-RNDP	278.063	6.867	0.000	95	291.102	35.754	0.086	93
VNS	279.250	36.560	0.160	68	292.020	33.890	0.100	71

Table 3: Comparison on the Barabási-Albert (B2) and Erdös-Rény (B3) benchmark sets from [23].

From the presented results, we further note that all the five algorithms exhibit a highly robust performance on the Barabási-Albert ($B2$) and Erdös-Rényi ($B3$) instances introduced in [23] (see boxplots (c) and (e)). Indeed, the normalized objective values are equal (or close) to zero, indicating the good ability of the algorithms to locate and reach the best-known solutions in most of the trials. Given a significant variation in performances of the five algorithms across the other benchmarks, this puts to question the hardness of the instances from [23]. Table 3 provides a comparison with the Variable Neighborhood Search (VNS) algorithm from [23] on these two sets of instances. For each algorithm, we report the average quality over all the instances (column 'Avg.'), the CPU time in seconds (column 'Time(s)'), the average percent deviation with respect to the best-known (or improved) so-

lution (column '% Dev.'), and the number of times that the given algorithm either matches or improves the best-known solution. Basically, BLS-RLE and the other four algorithms are able to improve the best result reported by VNS for 27/95 Barabási-Albert instances. For the Erdös-Rény instances, BLS-RLE, BLS and BLS-RND improve the result for 24/95, within a time that is generally less than a few seconds.

Benchmark set	BLS-RLE		BLS		BLS-RND		ILS-DIRP		ILS-RNDP	
	#best	*#avg*	*#best*	*#avg*	*#best*	*#avg*	*#best*	*#avg*	*#best*	*#avg*
B1 (17 instances)	15	15	1	2	1	0	0	0	0	0
B2 (180 instances)	156	134	123	95	157	147	161	82	96	96
B3 (140 instances)	138	135	132	119	138	136	133	81	121	114
B4 (35 instances)	33	26	15	13	16	10	5	0	0	0
B5 (50 instances)	54	51	41	20	43	24	31	16	3	1

Table 4: Frequency that each algorithm reports an equal or improved best and average performance on a given benchmark set, with respect to the competing algorithms.

Finally, the boxplot (h) sums up the performances of the five algorithms across all the benchmark instances. We may conclude that BLS-RLE shows the best overall performance, which is also confirmed by the statistical comparison from Table 2. The second best performing algorithm is BLS-RND, followed by BLS and ILS-DIRP. ILS-RND exhibits the worst performance and generally returns solutions with considerable deviation from the best-known solution. For each benchmark set and algorithm, Table 4 further presents the number of instances for which the given algorithm reports an equal or improved best (column '*#best*') and average (column '*#avg*') performance, with respect to the competing algorithms. These results confirm the success of BLS-RLE on instances from $B1$, $B4$ and $B5$, and show a slight advantage of BLS-RND over BLS-RLE on the $B2$ benchmark set.

For indicative purposes, the boxplot in Figure 3 presents the normalized times required by each algorithm to attain the reported average result, across all the benchmark instances. While BLS-RND and BLS-RLE require almost the same amount of time to reach the reported results (in terms of median and mean time values), BLS has only a slight advantage over BLS-RLE in terms of the median time value. Indeed, the proposed RL-based parameter controller introduces only a constant overhead in computational complexity compared to BLS, as action rewards, action values and action probabilities can be determined in constant time, if we exclude the time it takes to verify whether an attained local optimum is the hash table structure which is also done by BLS. Given the computed action probabilities, the selection of the next action for perturbation is performed in constant time, while the periodic sampling of action space has the complexity of $O(n)$ where n is the

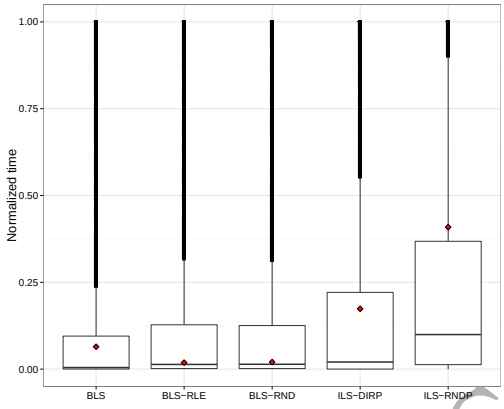


Figure 3: Boxplot of the normalized times, required by each algorithm to reach the reported average result, for all the benchmark instances.

size of P_{all}.

5. Analyses and Discussion

The purpose of this section is twofold. Firstly, the structure of the selected VSP instances is analyzed with the aim of determining the hardness of the used benchmark sets. Secondly, our aim is to provide the experiments to isolate the effects of important algorithmic components, with the purpose of understanding their behavior and evaluating their contribution to the success of the proposed approach.

The analyses are presented as follows. Section 5.1 provides analyses of the problem landscape to evaluate the hardness of the VSP benchmark sets and to show the structural properties of the considered instances. In Section 5.2.1, we analyze the effects of the pre-learning procedure and the amount of diversification introduced by each parameter pair setting. Section 5.2.2 further analyzes the time overhead introduced by the pre-learning procedure with the increase of the number of vertices and the graph density. Moreover, it investigates the possibility of a warm-start pre-learning procedure to eliminate or reduce the time overhead introduced by pre-learning. Finally, in Section 5.2.3, we evaluate the importance of the strategy for the action space sampling which is one of the distinguishing features of BLS-RLE.

5.1. Fitness-Distance Correlation and Distribution of Local Optima

The fitness-distance correlation analysis (FDC) provides useful indications about the problem hardness. It captures the correlation between the

solution fitness and its distance to the nearest best-known solution or global optimum (if available). For a minimization problem, if the solution fitness decreases with the decrease of the distance to the optimum, then the objective function should provide a good guidance for the search since there is a "path" to the optimum via solutions with decreasing (better) fitness. In case of perfect fitness-distance correlation, the fitness-distance correlation coefficient ρ takes a value close to 1. For correlation of $\rho \approx -1$, the fitness function is completely misleading. We consider an instance to be hard if $\rho \leq 0.250$. FDC can also be visualized with a fitness-distance plot, which graphically displays the data for computing the fitness-distance correlation coefficient ρ.

We analyze the landscape correlation and the distribution of local optima on 40 selected VSP instances, which is around 9% of our VSP benchmarks. The results reported for each instance are based on 2000 independent runs of BLS, and using hamming distance to measure the distance between two locally optimal solutions. Since the optimal solutions for the selected instances are not known, we use instead the best-known local optima and refer to them as global optima.

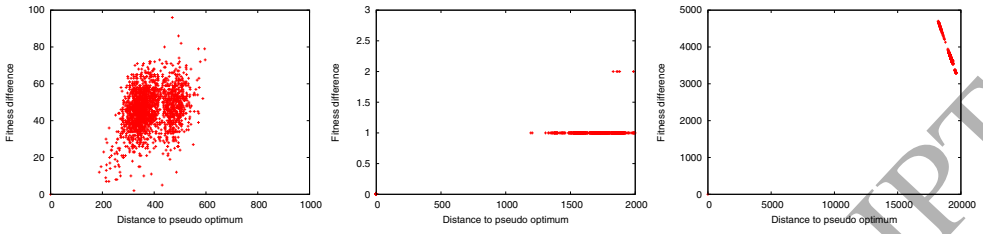
Table 5 summarizes the analytical results. Column 'ρ' shows a positive fitness-distance correlation in the problem landscapes, except for instances *c-43* and *soc-Epinions1* from the benchmark set $B4$. Therefore we consider these two instances as among the hardest ones used in our experiments, which might explain the low success rate of reaching the corresponding best-known solutions. For the other instances, we observe different degrees of landscape correlation with $0.142 \leq \rho \leq 0.987$, which implies that the used benchmark sets contain instances of various hardness. Characteristic examples are the *yeast* from B4 which is a quite challenging instance ($\rho = 0.142$), and $E4000\_003$, $E5000\_002$ from $B3$ which are among the easiest instances according to the value of ρ. The fitness-distance plot in Figure 4 visualizes the fitness-distance correlation for three selected VSP instance. These plots further show the difference in the landscape correlation between *soc-Epinions1* and $E5000\_002$. Another obvious indicator of the problem hardness, provided in Table 5, is the instance size (i.e., the number of vertices $|V|$, the number of edges $|E|$, as well as the graph density defined as $2|E|/|V|(|V|-1)$. In [14], the authors confirm that the hardness of the instances increases with the number of vertices and density. The graphs that we used in our experiments are significantly larger and thus much harder than those from the existing VSP benchmark set [14].

To gain a better insight into the distribution of locally optimal solutions in the search space, we further investigate the distances between locally

Table 5: Landscape analysis for 40 selected VSP instance from benchmark sets $B_1 - B_5$, based on 2000 independent runs of BLS. For each instance, we provide the number of vertices ($|V|$) and edges ($|E|$), the graph density, the average vertex degree (*avg. deg.*), the number of distinct local optima (i.e., the size of the analyzed sample, $\#lo$) and global optima found ($\#go$), the FDC coefficient (ρ), the average distance between a local optimum and its nearest local optimum (*avg.* $d_{\min}$), between local optima (*avg.* d_{all}), and between a local optimum and the nearest global optimum (*avg* d_{go}).

Instance	$\|V\|$	$\|E\|$	density	avg. deg.	#lo	#go	ρ	avg. $d_{\min}$	avg. d_{all}	avg. d_{go}
ibm02	19601	19584	0.000101	2.0	1991	1	0.455	301.3	460.5	388.4
ibm04	27507	31970	0.000085	2.3	1999	1	0.365	795.4	1069.4	986.4
ibm06	32498	34826	0.000066	2.1	1998	1	0.307	558.9	743.5	693.6
N2000_003_010	2000	5991	0.0030	6.0	1937	1	0.399	77.7	113.0	95.9
N3000_010_020	3000	29883	0.0066	19.9	1998	1	0.857	307.3	399.2	324.3
N3000_030_040	3000	89127	0.0198	59.4	2000	4	0.283	829.3	1161.0	1061.3
N4000_007_010	4000	27951	0.00340	14.0	2000	1	0.625	124.6	160.9	153.0
N4000_010_020	4000	39883	0.00498	19.9	1060	1	0.741	168.9	220.5	191.7
N4000_035_040	4000	138927	0.0174	69.5	1998	2	0.216	1236.6	1579.9	1492.5
N4000_045_050	4000	178259	0.0222	89.1	1998	11	0.316	1277.5	1619.8	1460.8
N5000_007_010	5000	69902	0.0280	14.0	2000	2	0.742	114.7	147.9	141.4
N5000_025_030	5000	124435	0.00995	49.8	2000	1	0.339	1112.9	1280.4	1149.5
E4000_001	4000	157385	0.0196	78.7	1998	5	0.472	1308.4	1660.5	1468.6
E4000_002	4000	313976	0.0393	157.0	1999	55	0.634	1243.6	1710.2	1358.2
E4000_003	4000	468183	0.0585	234.1	2000	250	0.972	1178.3	1739.2	1255.1
E5000_001	5000	245933	0.0197	98.4	1997	6	0.506	1699.5	2107.3	1886.5
E5000_002	5000	489837	0.0392	195.9	1998	622	0.987	1620.7	2168.4	1199.2
E5000_003	5000	730917	0.0580	292.4	1999	114	0.932	1487.6	2171.8	1738.1
c-43	11125	67400	0.00109	12.1	1999	2	-0.118	82.7	143.0	135.9
yeast	2361	7182	0.00258	6.1	2000	2	0.142	63.1	135.9	154.6
bcsstk17	10974	219812	0.00365	40.1	1100	164	0.638	18.2	292.0	209.3
ca-HepPh	7241	202194	0.00771	55.9	1991	1	0.496	708.8	866.8	1005.6
gre_1107	1107	5664	0.00925	10.2	897	1	0.393	3.9	39.3	152.8
sstmodel	3345	13047	0.00233	7.8	289	72	0.867	2.4	38.9	36.2
erdos992	6100	7515	0.00403	2.5	1953	2	0.526	65.4	97.8	86.4
lnsp3937	3937	25407	0.00328	12.9	1996	1972	0.838	44.3	93.1	0.7
oregon2_010505	5441	19505	0.00132	7.2	2000	1	0.661	27.5	47.0	34.8
email-Enron	9660	224896	0.00482	46.6	1998	3	0.845	547.1	771.9	606.3
soc-Epinions1	22908	389439	0.00148	34.0	2000	1	-0.030	1801.2	2577.6	19172.7
p2p-Gnutella05	8846	31839	0.00080	7.2	2000	1	0.449	1193.5	1342.9	1377.5
p2p-Gnutella06	8717	31525	0.00082	7.2	2000	1	0.401	1137.7	1297.8	1268.9
p2p-Gnutella08	6301	20777	0.00100	6.6	2000	1	0.481	646.9	759.2	732.3
G23	2000	19990	0.0100	20.0	1999	22	0.393	229.7	555.0	398.5
G25	2000	19990	0.0100	20.0	1997	35	0.515	188.1	478.9	317.0
G27	2000	19990	0.0100	20.0	1800	3	0.331	302.4	631.1	541.6
G30	2000	19990	0.0100	20.0	1991	30	0.504	279.4	616.3	435.7
G35	2000	11778	0.0589	11.8	2000	12	0.539	200.4	393.2	350.7
G36	2000	11778	0.0589	11.8	2000	1	0.371	209.1	432.8	414.1
G37	2000	11785	0.0589	11.8	2000	1	0.285	193.6	372.1	348.9
G42	2000	11779	0.0589	11.8	1860	3	0.267	228.1	429.3	409.7

optimal solutions from a given sample Q. Column '*avg.* $d_{\min}$' shows the average distance between a local optimum lo and its nearest local optimum lo' where $d_{\min}(lo) = \min_{lo' \in Q, lo' \neq lo} d(lo, lo')$, while columns '*avg.* d_{all}' and



(a) ibm02 (b) E5000_002 (c) soc-Epinions1

Figure 4: Fitness-distance correlation analysis (FDC) for 3 VSP instances.

'*avg.* d_{go}' show respectively the average distance between local optima from Q, and the average distance between a local optimum and the nearest global optimum. From Table 5 (columns '*avg.* $d_{\min}$' and '*avg.* d_{all}'), we observe that the distances between local optima are significant in most cases, considering the average vertex degree (see columns '*avg. deg*') that represents the average number of vertices affected by the utilized search operator. Only for few instances (e.g., *bcsstk*17, *sstmodel* and *gre*_1107), the minimal and average distances between local optima pairs are low. From column '#*lo*', we further observe that, for most instances, the search returns a new local optimum almost after each run. This implies that it should not be hard for the search to escape from an attraction basin of a local optimum. Moreover, relatively significant distances between local optima for most instances should prevent the search from cycling even with a weak diversification.

5.2. Analyses of the Algorithmic Components

5.2.1. Pre-learning Phase and the Parameter Pair Selection Mechanism

In this section, we analyze the outcome of the pre-learning phase and show the amount of diversification introduced by each parameter pair setting from $P_{all} = \{(l,e)_0, \ldots, (l,e)_n\}$. Based on the observations made from the analyses, we justify the selection choices for particular parameter pair settings $(l,e) \in P_{all}$ applied for perturbation.

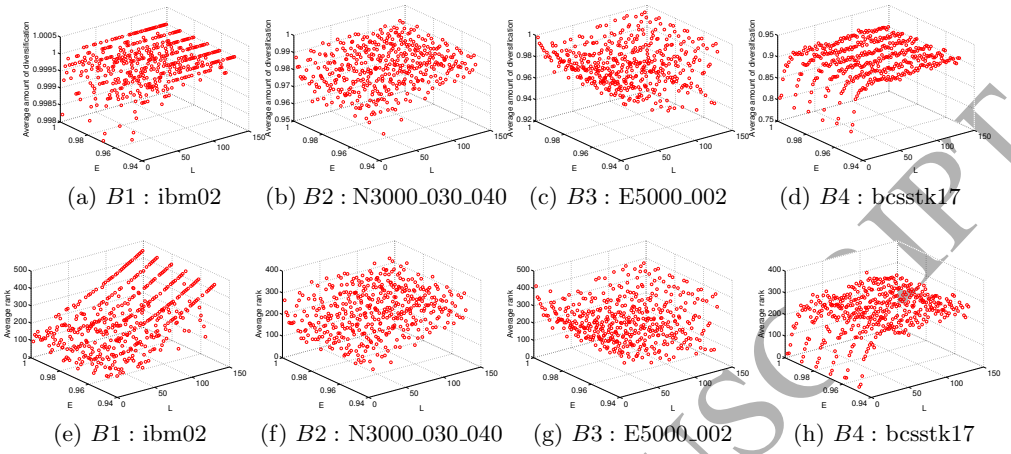
Because of the space limitations, we provide the results of the pre-learning analysis only for four selected instances (*ibm*02, *N*3000_030_040, *E*5000_002 and *bcsstk*17). For each instance, Figure 5 provides two 3D scatter plots that show respectively the average amount of diversification for each parameter pair setting $(l,e) \in P_{all}$ (top figures) and the average ranking for each $(l,e) \in P_{all}$ (bottom figures). For the sake of clarity, the amount of diversification is expressed as $\#new\_lo/\alpha$, where α is the number

of iterations of the pre-learning procedure per parameter pair setting and $\#new\_lo$ is the number of unique local optima encountered over α iterations. This analysis is based on 100 independent executions of the pre-learning procedure ($\alpha = 100$).

We observe from the top figures in Figure 5 that, in cases of *ibm*02, *N*3000_030_040, *E*5000_002, each pair (l, e) introduces on average a significant degree of diversification (i.e., not less than 0.92/1.00). This phenomenon may be justified by the distribution of local optima as observed in the previous section. Indeed, the distances between locally optimal solutions for these instances are quite significant, especially for *E*5000_002, which prevents the search from cycling given any setting of l and e. On the other hand, according to the information presented in Table 5 (see columns 'avg. $d_{\min}$' and 'avg. d_{all}'), local optima for *bcsstk*17 appear to be clustered together. This explains to some extent the lower degree of diversification introduced into the search for *bcsstk*17. As expected, plots for instance *ibm*02 and *bcsstk*17 also indicate that the degree of diversification increases with the increase of the number of perturbation moves l. On the other hand, the plots for *E*5000_002 and *N*3000_030_040 do not reveal such a clear correlation between (l, e) values and the amount of introduced diversification.

To show the deviation of the diversification degree from the mean for each $(l, e) \in P_{all}$, we provide error bars around the mean in Figure 6 (top figures) for the four selected instances. Moreover, the bottom figures in Figure 6 provides Standard Error (SE) of the mean rank for each $(l, e) \in P_{all}$. Despite the stochastic nature of the pre-learning procedure, we observe only a minor deviation of the diversification degree from the mean which implies a stable ranking (with a relatively small SE) assigned to each $(l, e) \in P_{all}$.

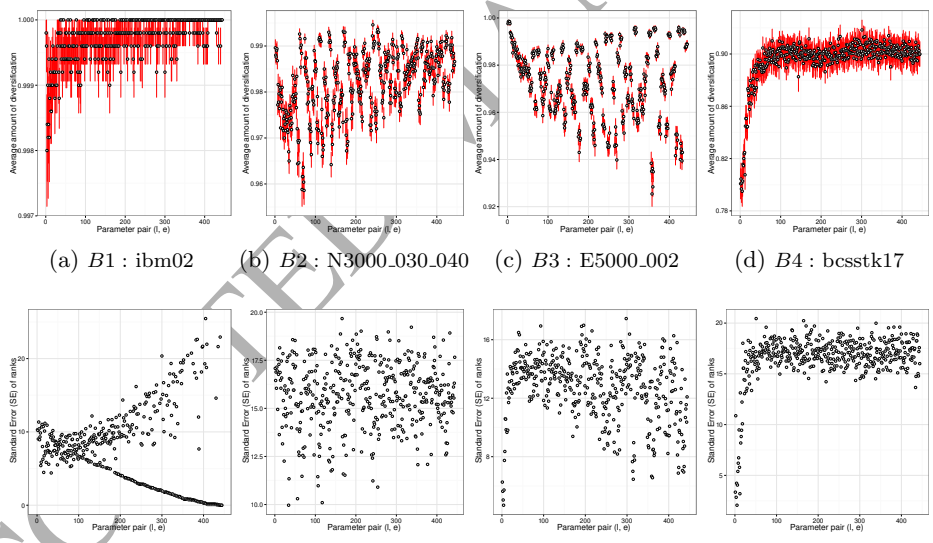
Finally, Figure 7 shows the selection and application frequency of each $(l, e)_i \in P_{all}$, $0 \leq i \leq n$ for the four selected VSP instances (*ibm*02, *N*3000_030_040, *E*5000_002 and *bcsstk*17). The x-axis indicates the rank i of $(l, e)_i \in P_{all}$, while the y-axis provides the normalized selection frequency of $(l, e)_i$. These results are based on one run of BLS-RLE with 500000 iterations of the main procedure, using the default parameter settings. For instances *ibm*02, *N*3000_030_040 and *E*5000_002, we observe that BLS-RLE is biased towards parameter pair settings with a lower rank since they introduce an amount of diversification sufficient to encounter new local optima. On the other hand, in case of *bcsstk*17, parameter pairs of higher ranks are being selected more frequently since lower ranked pairs are often not sufficient to escape from local optima in the given landscape. This behavior thus coincides well with the observations made from our previous analyses, where we observed low minimal and average distances between local optima



(a) $B1$: ibm02 (b) $B2$: N3000_030_040 (c) $B3$: E5000_002 (d) $B4$: bcsstk17

(e) $B1$: ibm02 (f) $B2$: N3000_030_040 (g) $B3$: E5000_002 (h) $B4$: bcsstk17

Figure 5: 3D scatter plots showing the average degree of diversification introduced with each $(l, e) \in P_{all}$ (top figures) and the average rank for each $(l, e) \in P_{all}$ (bottom figures).



(a) $B1$: ibm02 (b) $B2$: N3000_030_040 (c) $B3$: E5000_002 (d) $B4$: bcsstk17

(e) $B1$: ibm02 (f) $B2$: N3000_030_040 (g) $B3$: E5000_002 (h) $B4$: bcsstk17

Figure 6: Error bars showing the variability of diversification degree for each $(l, e) \in P_{all}$ (top figures) and standard error of ranks (bottom figures).

in case of *bccsstk*17, and significantly higher distances between local optima for *ibm*02, *N*3000_030_040 and *E*5000_002 (see Table 5).

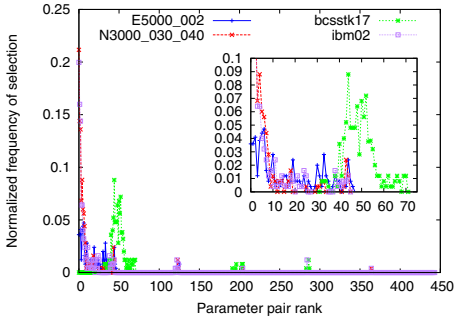


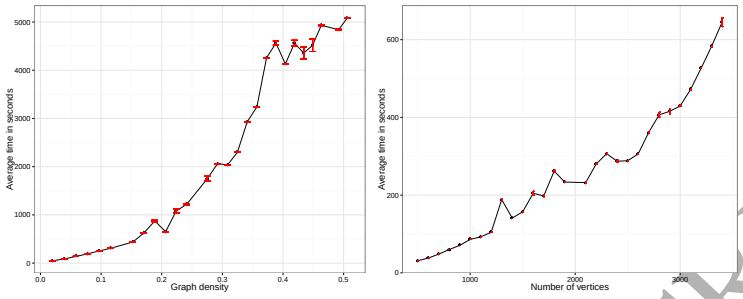
Figure 7: Frequency of selection and application of parameter pair settings $(l,e)_i \in P_{all}$ for the 4 selected VSP instances.

5.2.2. Time Overhead and Warm-Start of the Pre-learning Procedure

This section shows the time overhead incurred by the pre-learning phase. Moreover, we investigate the possibility of a warm-start pre-learning procedure, i.e., reuse of the information (parameter pair ranks) learned in this phase for a different instance with same or similar properties.

Since it was confirmed in [14] that the hardness (complexity) of a VSP instance is highly correlated with the number of vertices and the graph density, it is reasonable to assume that the time required for the pre-learning phase depends on these two factors. To demonstrate this point, we generate two benchmark sets of 30 instances with the Erdös-Rényi model [47]. To generate the first benchmark set, we fix the number of vertices to 2000 and use increasing values (from 0.01 to 0.30) for the probability p of connecting a pair of vertices. The second benchmark set includes graphs with $|V| = \{500, 600, 700, \ldots, 3400\}$ and densities of approximately 0.096. The plots in Figure 8 illustrate the overhead time incurred by the pre-learning procedure for the first and the second set of benchmark instances. The x-axis corresponds to instances in order of increasing densities (number of vertices), while the y-axis shows the average time in seconds required by the pre-learning phase over 30 independent executions, with $\alpha = 100$. As expected, we observe a significant increase in the time overhead with the increase of the graph density (sub-figure (a)) as well as with the increase of the number of vertices (sub-figure (b)).

We next investigate the possibility of reducing or eliminating this time overhead by using parameter pair ranks determined for an instance of the same or similar structure, from the same source (benchmark). Therefore, we use the parameter pair ranks determined for instances ibm01, N2000_003_010,



(a) (b)

Figure 8: Pre-learning time overhead with the increase of the graph density (sub-figure (a)) and the number of vertices (sub-figure (b)).

E1000_001_01, and G1 when running our algorithm on benchmark sets $B1, B2, B3$ and $B5$ respectively. In Figure 9, we provide boxplot graphs to illustrate the impact of a warm-start pre-learning procedure on solution quality, when combined with BLS-RLE. For each benchmark set ($B1$, $B2$, $B3$ and $B5$), we compare the normalized objective values obtained with the standard BLS-RLE, and with those obtained with the BLS-RLE using a warm-start pre-learning procedure. Figure 9 also reports p-values, based on the Wilcoxon test, for the mean results reported by the two BLS-RLE versions. From Figure 9, we observe a negative impact of the warm-start pre-learning procedure only in case of the benchmark set $B5$, with $p < 0.0001$. Surprisingly, BLS-RLE with warm-start pre-learning performs significantly better than the standard BLS-RLE for the benchmark set $B2$ ($p < 0.0001$), and slightly better than the standard BLS-RLE for instances from $B1$ ($p = 0.517$). Notice that the benchmark instances in the $B1, B2$, and $B3$ sets belong to the same classes of problems, while $B5$ is a collection of three different types of graphs (toroidal, planar, and random graphs). As such, these results imply that a warm-start pre-learning mechanism could be considered with BLS-RLE to reduce the overhead time incurred by the pre-learning phase, especially for problem instances with same characteristics.

5.2.3. Analysis of the Action Space Sampling Strategy

One of the key features of the proposed algorithm is the dedicated strategy for action space sampling (see Section 3.5), which enables rapid convergence towards a limited set of actions that appear to be the most convenient for the given state of search. To evaluate the impact of this element to the overall success of BLS-RLE, we compare BLS-RLE with the following mod-

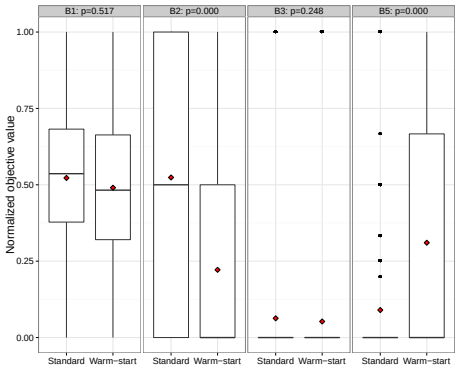


Figure 9: Boxplots comparing the normalized objective values for the benchmark sets $B1$, $B2$, $B3$ and $B5$, obtained by the standard BLS-RLE (without warm-start pre-learing) and the BLS-RLE using a warm-start pre-learning procedure.

ifications of our algorithm:

- **BLS-RLE-Pall:** Unlike BLS-RLE, this algorithm does not perform any sampling of the actions. Instead, it allows selection of any action from the set of all the possible actions P_{all} at any given time. As such, the learning is performed on all the actions from P_{all}, i.e., $P_{learn} = P_{all}$;
- **BLS-RLE-Pall-restart:** This algorithm is a slight modification of BLS-RLE-Pall with a restart mechanism, which resets action values corresponding to each $(l, e) \in P_{all}$ to 1 after ϵ time steps ($\epsilon = 2000$);
- **BLS-RLE-RND:** This algorithm performs sampling of the action space as in BLS-RLE. However, BLS-RLE-RND employs a different strategy to update P_{learn} after ϵ time steps, which consists of replacing the worst performing action from P_{learn} with a randomly selected action from $P_{all} \setminus P_{learn}$.

We carry out the comparison using the selected instances shown in Table 5 (30 independent runs, with a maximal time limit set to 2000 seconds), and the default parameter values (as presented in Table 1). To illustrate the performance of the four algorithms, Figure 10 presents a box-plot graph of their normalized performances over all the considered problem instances and executions. It can be clearly observed that BLS-RLE has a performance superior to those of the reference algorithms, which is confirmed by the p-values from both the Wilcoxon-signed rank test and the Bonferroni correction method provided in Table 6 (a statistically significant difference between the algorithms was verified beforehand by a Friedman test, $p < 0.001$ and

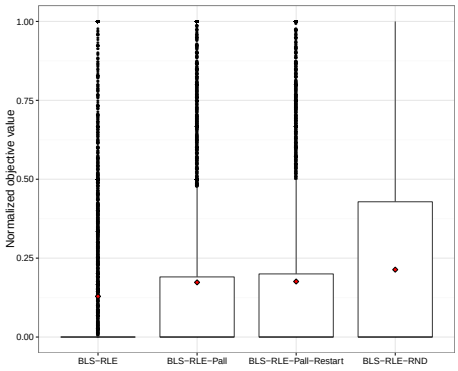


Figure 10: Boxplot of the normalized objective values obtained with BLS-RLE, BLS-RLE-Pall, BLS-RLE-Pall-restart and BLS-RLE-RND algorithms over all the considered problem instances and runs.

	BLS-RLE-Pall		BLS-RLE-Pall-restart		BLS-RLE	
	p	p_{bonf}	p	p_{bonf}	p	p_{bonf}
BLS-Pall-restart	**0.045**	0.272	–	–	–	–
BLS-RLE	**<0.001**	**<0.001**	**<0.001**	**<0.001**	–	–
BLS-RLE-RND	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**

Table 6: Post-hoc analysis of the normalized performances reported with BLS-RLE and its competitors across all the benchmark sets.

$\chi^2 = 96.891$). Indeed, we observe that the normalized objective values obtained with BLS-RLE generally go to zero. According to the boxplot and the p-values from Table 6, the second best performance is obtained with BLS-RLE-Pall and BLS-RLE-Pall-restart which perform equally well, followed by BLS-RLE-RND. These results not only emphasize the importance of action space sampling for BLS-RLE but also highlight the significance of the proposed action space sampling strategy used by BLS-RLE (see Section 3.5).

6. Conclusion

This paper presented an improved Breakout Local Search (BLS) approach, based on ideas that draw upon reinforcement learning, for the vertex separator problem. As the previous BLS algorithms, the proposed BLS method (denoted as BLS-RLE) iterates alternatively between a descent-based local search to reach a local optimum and an adaptive multi-type perturbation to escape from the attained local optimum. The main feature of BLS-RLE is a novel parameter controller that interdependently determines the values for two variable-parameters, the number l of perturbation

moves and the probability e of selecting tabu-based over random perturbation, depending on the success of the given parameter setting in the previous perturbation phases. More precisely, to escape from a local optimum s, BLS-RLE selects a parameter pair setting (l, e) from a limited learning set P_{learn} according to its action value, which is determined using a Softmax rule based on an empirical quality estimate of the given parameter pair setting. The originality and success of the proposed parameter controller lie in the two following key elements: (i) a dedicated strategy for action space sampling, which enables rapid convergence towards a limited set of actions that appear to be the most convenient for the given state of search (ii) a reward function which complies with the principle "intensification first, minimal diversification only if needed".

To evaluate the ability of the proposed method to self-adaptively establish the most suitable balance between intensification and diversification, we performed extensive computational experiments on a wide set of instances with different sizes and structures. A large number of these instances is motivated by several VSP applications including VLSI design, cyber security, and decoy routing. The obtained results highlight the benefit of the proposed parameter control mechanism. Indeed, BLS-RLE significantly improves the performance of BLS and competes very favorably with several other variations of ILS. Given the generality of the proposed RL-based parameter controller, it could be used with BLS for other important combinatorial problems. The parameter controller is available online for future use, and its success on other problems is going to be the topic of our future work.

Another contribution of this work are analyses of the problem landscape that aim to evaluate the hardness of the used benchmark instances, and to explain the behavior of BLS-RLE. Moreover, we justified the complexity of the method with detailed analyses of the key algorithmic elements. A careful computational complexity analysis of the proposed approach will be studied in a future work.

Acknowledgment

This work is funded by the Engineering and Physical Sciences Research Council (EP/J017515/1) and the NSFC Research Fund for International Young Scientists (no. 61450110443). M.G. Epitropakis is also funded by the Lancaster University Early Career Internal Grant (A100699).

References

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

[2] C. Leiserson, Area-efficient graph layout (for vlsi), in: Proceedings of the 21st Annual Symposium on the Foundations of Computer Science, IEEE Press, 1980, pp. 270–281.

[3] J. Ullman, Computational Aspects of VLSI, Computer Science Press, Rockville, 1984.

[4] B. Fu, Z. Chen, Sublinear time width-bounded separators and their application to the protein side-chain packing problem, in: C. P. S.W. Cheng (Ed.), Algorithmic Aspects in Information and Management, Vol. 4041 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 149–160.

[5] K. Donghyun, G. Frye, K. Sung-Sik, J. Hyung, A. Tokuta, On combinatoric approach to circumvent internet censorship using decoy routers, in: Military Communications Conference, MILCOM 2013 - 2013 IEEE, 2013, pp. 593–598.

[6] M. Schuchard, J. Geddes, C. Thompson, N. Hopper, Routing around decoys, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, ACM, New York, NY, USA, 2012, pp. 85–96.

[7] M. Caria, A. Jukan, M. Hoffmann, SDN Partitioning: A Centralized Control Plane for Distributed Routing Protocols, IEEE Transactions on Network and Service Management 13 (3) (2016) 381–393, arXiv: 1604.04634.

[8] S. N. Bhatt, F. T. Leighton, A framework for solving VLSI graph layout problems, Journal of Computer and System Sciences 28 (2) (1984) 300 – 343.

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

[10] E. Kayaaslan, A. Pinar, Ü. Çatalyürek, C. Aykanat, Partitioning hypergraphs in scientific computing applications through vertex separators on graphs, SIAM Journal on Scientific Computing 34 (2) (2012) A970–A992.

[11] R. Lipton, R. Tarjan, A separator theorem for planar graphs, SIAM Journal on Numerical Analysis 36 (1979) 177–189.

[12] J. Liu, A graph partitioning algorithm by node separators, ACM Transactions on Mathematical Software 15 (3) (1989) 198–219.

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

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

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

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

[17] U. Feige, M. Hajiaghayi, J. Lee, Improved approximation algorithms for minimum-weight vertex separators, in: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, STOC '05, ACM, New York, NY, USA, 2005, pp. 563–572.

[18] U. Feige, M. Mahdian, Finding small balanced separators, in: Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing, STOC '06, ACM, New York, NY, USA, 2006, pp. 375–384.

[19] W. Hager, J. Hungerford, Continuous quadratic programming formulations of optimization problems on graphs, European Journal of Operational Research 240 (2) (2015) 328 – 337.

[20] A. George, J. W. H. Liu, An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems, SIAM Journal on Numerical Analysis 15 (5) (1978) 1053–1069.

[21] J. W. H. Liu, A Graph Partitioning Algorithm by Node Separators, ACM Transactions on Mathematical Software 15 (3) (1989) 198–219.

[22] U. Benlic, J. Hao, Breakout local search for the vertex separator problem, in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, 2013, pp. 461–467.

[23] J. Sánchez-Oro, N. Mladenović, A. Duarte, General Variable Neighborhood Search for computing graph separators, Optimization Letters (2014) 1–21.

[24] H. Lourenco, O. Martin, T. Stützle, Iterated local search, Handbook of Metaheuristics, Springer-Verlag, Berlin Heidelberg, 2003.

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

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

[27] R. Sutton, A. Barto, Introduction to Reinforcement Learning, 1st Edition, MIT Press, Cambridge, MA, USA, 1998, 02767.

[28] S. Müller, N. Schraudolph, P. Koumoutsakos, Step size adaptation in evolution strategies using reinforcement learning, in: Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC '02., Vol. 1, 2002, pp. 151–156.

[29] A. E. Eiben, M. Horvath, W. Kowalczyk, M. C. Schut, Reinforcement learning for online control of evolutionary algorithms, in: Proceedings of the 4th International Conference on Engineering Self-organising Systems, ESOA'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 151–160.

[30] G. Karafotias, M. Hoogendoorn, A. E. Eiben, Parameter Control in Evolutionary Algorithms: Trends and Challenges, IEEE Transactions on Evolutionary Computation 19 (2) (2015) 167–187.

[31] R. Battiti, M. Brunato, P. Campigotto, Learning while optimizing an unknown fitness surface, in: V. Maniezzo, R. Battiti, J. Watson (Eds.), Learning and Intelligent Optimization, Vol. 5313 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 25–40.

[32] S. Prestwich, Tuning local search by average-reward reinforcement learning, in: V. Maniezzo, R. Battiti, J. Watson (Eds.), Learning and Intelligent Optimization, Vol. 5313 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 192–205.

[33] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2-3) (2002) 235–256.

[34] H. Robbins, Some aspects of the sequential design of experiments, Bulletin of the American Mathematical Society 5 (58) (1952) 527–535.

[35] R. Battiti, M. Brunato, F. Mascia, Reactive Search and Intelligent Optimization, Vol. 45 of Operations research/Computer Science Interfaces, Springer Verlag, 2008.

[36] S. Ghandi, E. Masehian, A breakout local search (bls) method for solving the assembly sequence planning problem, Engineering Applications of Artificial Intelligence 39 (0) (2015) 245 – 266.

[37] A. Fialho, Adaptive operator selection for optimization, Ph.D. thesis, Université Paris-Sud XI, Orsay, France (December 2010).

[38] M. Epitropakis, D. Tasoulis, N. Pavlidis, V. Plagianakos, M. Vrahatis, Tracking differential evolution algorithms: An adaptive approach through multinomial distribution tracking with exponential forgetting, in: I. Maglogiannis, V. Plagianakos, I. Vlahavas (Eds.), Artificial Intelligence: Theories and Applications, no. 7297 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 214–222.

[39] D. Thierens, Adaptive strategies for operator allocation, in: F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, no. 54 in Studies in Computational Intelligence, Springer Berlin Heidelberg, 2007, pp. 77–90, 00042.

[40] T. Bäck, D. B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press, 1997.

[41] F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Vol. 54 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2007.

[42] M. Epitropakis, V. Plagianakos, M. Vrahatis, Evolutionary adaptation of the differential evolution control parameters, in: IEEE Congress on Evolutionary Computation, 2009. CEC '09, 2009, pp. 1359–1366.

[43] M. Epitropakis, D. Tasoulis, N. Pavlidis, V. Plagianakos, M. Vrahatis, Tracking particle swarm optimizers: An adaptive approach through multinomial distribution tracking with exponential forgetting, in: 2012 IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1–8.

[44] G. Karafotias, M. Hoogendoorn, A. Eiben, Why parameter control mechanisms should be benchmarked against random variation, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, 2013, pp. 349–355.

[45] R. Albert, A. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47–97.

[46] R. Albert, H. Jeong, A. Barabási, Error and attack tolerance of complex networks, Nature 406 (6794) (2000) 378–382.

[47] P. Erdös, A. Rényi, On random graphs. i, Publicationes Mathematicae 6 (1959) 290–297.

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

[49] M. Hollander, D. Wolfe, E. Chicken, Nonparametric Statistical Methods, 3rd Edition, Wiley, 2013.