A reinforcement learning-guided breakout local search algorithm for the heaviest subgraph problem

He Zheng[a,b], Jin-Kao Hao[a,*], Rafael Martí[b], Yang Wang[c,*]

[a] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers, 49045, France*
[b] *Departamento de Estadística e Investigación Operativa, Universitat de València, Dr. Moliner 50, València, 46100, Spain*
[c] *School of Management, Northwestern Polytechnical University, 127 Youyi West Road, Xi'an, 710072, China*

Computers and Operations Research, 2026, https://doi.org/10.1016/j.cor.2026.107674

Abstract

We propose a reinforcement learning-guided breakout local search algorithm for the Heaviest k-Subgraph Problem, a general optimization graph model with many applications. This algorithm combines a powerful local search procedure with a learning-guided perturbation strategy to achieve an effective balance between intensification and diversification in the search for high-quality solutions. The local search procedure explores two neighborhoods reinforced by a neighborhood reduction strategy. The learning-guided perturbation strategy uses Q-learning to select the most promising perturbation during the search. Computational results on benchmark instances show that the algorithm competes very favorably with the state-of-the-art approaches. It provides improved best results (new lower bounds) for 128 out of 177 instances and matches the best-known solutions for most remaining instances. The algorithm's competitive performance on three related problems (Densest k-Subgraph, Maximum Clique, and Hidden Clique) demonstrates its potential for broader applicability.

Keywords: Combinatorial optimization; Heaviest subgraph problem; Maximum diversity problem; Learning-guided perturbation; Heuristics.

*Corresponding author
Email addresses: hzzheng65@gmail.com (He Zheng), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), rafael.marti@uv.es (Rafael Martí), yangw@nwpu.edu.cn (Yang Wang)

1. Introduction

Let $G(V, E)$ be an undirected graph, where V is a set of vertices with $|V| = n$ and E is a set of edges, let $w : E \rightarrow R^+ \cup \{0\}$ be an edge weighting function, which associates a non-negative value w_{ij} to each edge $\{i, j\} \in E$. The Heaviest k-Subgraph Problem (HSP) is to determine a subset U of k vertices (k is given) such that the total edge weight of the k-subgraph induced by U is maximized. Formally, the problem can be stated as a quadratic 0-1 formulation:

$$Maximize\ f(S) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} x_i x_j \tag{1}$$

$$s.t. \sum_{i=1}^{n} x_i = k \tag{2}$$

$$x_i \in \{0, 1\}, i = 1, \ldots, n \tag{3}$$

where x_i is a binary variable indicating whether vertex i belongs to the k-subgraph.

Given the variety of the contexts where HSP finds applications, we can identify several variations of the original model, addressed by Macambira (2002) as the maximum edge-weighted subgraph problem. In particular, HSP generalizes a classic problem in graph theory, the Densest k-Subgraph (DkS) (Kortsarz and Peleg, 1993) that limits itself to graphs with edge weights equal to one. Similarly, in the Maximum Diversity Problem (MDP) (Kuby, 1988), also called the maxsum problem (Martí et al., 2022), the edge weights are distances satisfying the triangle inequality, making MDP another special case of HSP. Under the classical MDP assumptions, the graph is complete, meaning every pair of points is connected with an edge weighted by their distance. Selecting a subset that maximizes the sum of distances is thus equivalent to finding the heaviest subgraph. In contrast, HSP allows incomplete graphs, where only existing edges contribute to the objective. Applying HSP to a complete graph with distances as edge weights reduces it to MDP, so every classical MDP instance can be viewed as a special case of HSP.

HSP is known to be useful in a variety of real-world applications. Examples in-

clude the detection of communities or link spams in social networks (Gibson et al., 2005; Saarinen et al., 2023), the identification of real-time stories (Sotirov, 2020), the extraction of molecular structure, and the facility location (Prokopyev et al., 2009). HSP is also a relevant model to identify a group of customers with the highest total traffic flow in telecommunication networks (Brimberg et al., 2009), and to find a team of employees with the highest collaborative compatibility (Gajewar and Das Sarma, 2012). Meanwhile, as an NP-hard problem, solving HSP is computationally challenging (Corneil and Perl, 1984).

Given the interest of HSP, a number of solution methods have been investigated to solve it. As will be shown in Section 2, existing studies on HSP (and DkS) have mainly focused on exact or approximation algorithms, and they cannot efficiently deal with large-scale instances, particularly those with thousands of vertices and large selection sizes. While some metaheuristic approaches have been explored to tackle these computational challenges, they often struggle with limited search ability and insufficient mechanisms to escape from local optima.

To address these limitations, we turn to the Breakout Local Search (BLS) framework, which has shown strong performance on several challenging problems (Benlic and Hao, 2013a,b,c; Ghandi and Masehian, 2015; El Krari et al., 2020; He et al., 2022). BLS emphasizes the importance of an adaptive perturbation mechanism, which aims to go beyond encountered local optima while promoting diversification. Combined with an efficient descent-based local search, BLS implements a balance between intensification and diversification, making it an effective metaheuristic suitable to deal with challenging instances of HSP.

The perturbation mechanism in conventional BLS exploits information related to the search state and progress to adaptively determine the degree of diversification. While effective in many cases, it may not achieve the necessary level of diversification in a complex search landscape. Reinforcement Learning (RL) techniques (Sutton and Barto, 1998) offer advanced alternatives for decision-making strategies from experience to adaptively guide choices during the search (Bengio et al., 2021; Angioni et al., 2025). Building on this idea, Benlic et al. (2017) used the Softmax action-selection rule (Sutton and Barto, 1998) to interdependently determine the values of

a parameter pair: the number of perturbation moves and the probability of selecting one perturbation type over another. However, learning in a large space of parameter pair values is challenging.

In this work, we propose a Reinforcement Learning-guided Breakout Local Search (RL-BLS) algorithm. Our goal is twofold: (1) to provide a superior solution method for HSP; and (2) to investigate and demonstrate how RL techniques can fundamentally enhance the BLS framework itself, resulting in a more robust method. This contributes to research on the intersection of optimization techniques and machine learning. Our contributions are summarized as follows.

- The proposed RL-BLS algorithm integrates complementary search strategies to achieve a good balance between intensification and diversification. It performs effective local optimization using the popular *Swap* operator and the novel *SwapPair* operator, enhanced by a newly proved neighborhood reduction strategy that preserves the most promising moves. For diversification, its Q-learning-guided perturbation selection mechanism dynamically selects the most promising perturbation type from four options of varying intensities.

- We evaluate RL-BLS on the 177 popular HSP benchmark instances. The results show that RL-BLS competes very favorably with the state-of-art methods. In particular, it achieves new best objective values (improved lower bounds) for 128 instances (for more than 72% of the instances) and matches the best-known results on all but one remaining instance. We also show the competitiveness of the algorithm for solving the related MDP problem.

The rest of the paper is organized as follows. After a literature review on the works related to HSP in Section 2, we present in Section 3 the proposed RL-BLS algorithm, including its general framework, the constrained neighborhoods, and the Q-learning-guided perturbation selection. Section 4 shows the comparative results on benchmark instances and Section 5 investigates the effectiveness of the algorithm's key components. Finally, Section 6 provides concluding remarks.

2. Literature Review

HSP and its unweighted variant have been shown to be NP-hard in the early work of Corneil and Perl (1984), which is often cited as one of the earliest references related to this problem. Early studies, such as the work of Billionnet (2005), have focused on mathematical formulations of HSP. Several later works have proposed dedicated exact or approximation approaches to solve HSP and its unweighted version DkS. In particular, Letsios et al. (2016) designed a branch-and-bound algorithm that is able to compute exact solutions for large instances in social networks. Furthermore, the authors also developed an approximated version of the algorithm to solve instances with larger k values (up to 105). Krislock et al. (2016) used the branch-and-bound method combined with an improved semidefinite bounding procedure to solve DkS. Numerical results show that it can solve problem instances with up to 160 vertices.

Singh et al. (2019) introduced an approximation algorithm for HSP, which integrates a greedy graph reduction strategy with a branch-and-bound procedure. Results show that the algorithm can solve large graphs with k larger than 19, and the approximation ratio to optimal solutions is consistently above 0.9. Sotirov (2020) introduced a coordinate descent approximation approach, with a particular focus on the simultaneous update of many coordinates. The method is applied to both HSP and DkS instances. Its results on the tested graphs are shown to be at most 2% worse than the best-known solutions obtained by heuristics.

To achieve superior results for large graphs in a reasonable time for HSP, several heuristic and metaheuristic algorithms have been studied. Macambira (2002) described a tabu search (TS) heuristic for HSP, which uses three strategies to obtain an initial solution, and the swap neighborhood for local optimization. The method has remarkable results compared to previous methods. Brimberg et al. (2009) presented a variable neighborhood search (VNS) heuristic for solving HSP. The authors examined the performance of several versions of the heuristic, including basic VNS (BVNS) and skewed VNS (SVNS), as well as a combination of two greedy constructive heuristics. A set of 177 large and hard HSP instances were also introduced. Extensive experiments on these instances show that BVNS exhibits a superior per-

formance compared to other heuristic approaches that were evaluated. Saarinen et al. (2023) improved the neighborhood change and search schemes of BVNS, and proposed an opportunistic Variable Neighborhood Search (OVNS) algorithm that takes advantage of empirical insights about social network structures. Their experimental results illustrate that OVNS outperforms BVNS on network graphs.

One notices that heuristic methods are more commonly used for solving the tightly related MDP. Early heuristic techniques include iterated local search (ITS) (Palubeckis, 2007), greedy randomized adaptive search procedure (GRASP) (Duarte and Martí, 2007), scatter search (SS) (Gallego et al., 2009), and others. Most of later works on MDP have been conducted based on the memetic search framework (MA). Wu and Hao (2013) presented the first memetic algorithm (MAMDP) for MDP, which integrates a constrained neighborhood tabu search procedure with a population-based evolutionary algorithm. The results demonstrate excellent performance, yielding numerous best-known solutions for large MDP instances. Wang et al. (2014) devised an improved version of MAMDP for MDP, called TS/MA algorithm, which applies a successive filter candidate list strategy for the neighborhood exploration and a solution property-based combining strategy. Their results indicate the competitiveness of the strategies introduced. Zhou et al. (2017) proposed an opposition-based memetic algorithm (OBMA) for solving MDP, which integrates the concept of opposition-based learning (OBL) into memetic search, together with an improved parametric constrained neighborhood. This powerful algorithm updated many best-known solutions on MDP benchmark sets. Other heuristics based on MA include the memetic self-adaptive evolution strategy (MSES) (De Freitas et al., 2014), the diversification-driven memetic algorithm (DMAMDP) (Lai et al., 2018), and the two-phase tabu search-based evolutionary algorithm (TPTS/EA) (Liu et al., 2022). For a comprehensive review and comparison of these algorithms for MDP, we refer the reader to Martí et al. (2022); Martí and Martínez-Gavara (2023).

To the best of our knowledge, the current state-of-the-art heuristics for solving HSP are BVNS by Brimberg et al. (2009) and OVNS by Saarinen et al. (2023). For MDP, OBMA by Zhou et al. (2017) outperforms earlier methods and some later MDP heuristics have similar performances compared to OBMA.

3. Reinforcement Learning-guided Breakout Local Search for HSP

3.1. General Procedure

As an iterated local search method, BLS (Benlic and Hao, 2013b,c) alternates between two core phases: a descent-based local search phase to intensify the search in a given region of the solution space, and an adaptive perturbation phase to escape from local optima. The local search phase explores one or several neighborhoods to find high quality local optima in a particular search region. Of particular interest is the perturbation phase, which aims to reach an appropriate degree of diversification at different search stages using a self-adaptive strategy. This adaptive strategy is mainly achieved by changing the type of perturbation and the number of perturbation moves (i.e., jump magnitude) applied according to the search history. In BLS, the former is achieved by determining the probability of selecting one perturbation type over others. Our RL-BLS algorithm enhances this fundamental BLS framework by using Q-learning (Watkins and Dayan, 1992) to intelligently manage the task of selecting the applied perturbation during this diversification phase.

Algorithm 1 presents the pseudocode of the RL-BLS algorithm. Starting from an initial solution S (line 1), the best solution S^* and several parameters are initialized, including the jump magnitude L, the average deviation $AvgD$, and the RL components (lines 2–6). The algorithm then iteratively alternates between local search and perturbation (lines 7–22). In the local search phase, the Constrained Neighborhood Local Search $CNLS$ is applied to reach a local optimum (line 8). The current state st is then identified, and the RL module updates the reward records and Q-table accordingly (lines 9–13). If S improves upon S^*, the best solution is updated (lines 15–17). In the perturbation phase, the Q-learning module selects the most suitable perturbation type T (line 18) and the jump magnitude L is adaptively determined from the search history (line 19). The solution is then perturbed with (L, T) (line 20), yielding a new starting point for the next round of local search. This process continues until a stopping condition (e.g., a cutoff time or maximum number of iterations) is met, after which the best solution S^* is returned (line 23).

Algorithm 1 Reinforcement Learning-guided Breakout Local Search for HSP

Input: An edge-weighted graph $G = (V, E)$, and an integer $k < n$ ($n = |V|$), the minimal jump magnitude L_{min}
Output: The best solution S^* found

$S \leftarrow Initialize()$ /* Generate an initial solution */
$S^* \leftarrow S$, $f(S^*) = f(S)$ /* Record the best solution found and its objective value */
$L = L_{min}$ /* Set initial jump magnitude */
$AvgD = 0$ /* Record the average deviation between the local optimal solution and the optimal solution in all previous rounds */
$\mathcal{R}, \mathcal{Q} \leftarrow 0$ /* Initialize the Q-learning functions */
$(st', T') \leftarrow \emptyset$ /* Record the state and perturbation type */
while the stop condition is not met **do**
  $S \leftarrow CNLS(AvgD, S)$ /* Search in two constrained neighborhoods induced by $Swap$ and $SwapPair$ */
  $st \leftarrow DetermineState(S, HT)$
  if previous state-action pair (st', T') exists **then**
    $\mathcal{R}(st', T') \leftarrow UpdateReward(st, st', T')$
    $\mathcal{Q}(st', T') \leftarrow UpdateQtable(\mathcal{R}, \mathcal{Q}, st', T')$
  end if
  Update the hash table HT and the average deviation $AvgD$
  if $f(S) > f(S^*)$ **then**
    $S^* \leftarrow S$, $f(S^*) = f(S)$
  end if
  $T \leftarrow DetermineT(\mathcal{R}, \mathcal{Q}, st)$ /* Determine the best possible perturbation type in current state st using Q-learning */
  $L \leftarrow DetermineL(S, L, HT)$ /* Determine the jump magnitude using historical information */
  $S \leftarrow Perturb(S, L, T)$
  $(st', T') \leftarrow (st, T)$
end while
return S^*

3.2. Initial Solution

For a given graph $G(V, E)$ and an integer k, a candidate solution S (equivalently the subset of the selected vertices U) is represented by a vector of length n, $S = \{x_1, x_2, ..., x_n\}$, where $x_i = 1$ if vertex i is in the subset U of the selected vertices; $x_i = 0$ otherwise. For each vertex $i \in V$, its weight contribution to the objective value is recorded in α_i (Brimberg et al., 2009; Wu and Hao, 2013; Zhou et al., 2017),

which is defined as the total weight of the edges from i to all the selected vertices, i.e., $\alpha_i = \sum_{j \in U} w_{ij}$. The weight contribution vector α is created when constructing an initial solution and is updated after each move.

In order to obtain an initial solution S of reasonable quality, the following greedy procedure is applied. At the beginning, all the vertices in V are included in the solution. Based on this solution S, the α_i value for each vertex i is computed for the first time. Then vertices in U are iteratively dropped one by one until U contains exactly k vertices. This solution construction procedure is commonly referred to in the heuristic literature as a "destructive method" since, unlike constructive methods that add one element at a time, this approach removes one element in each iteration. The interest of such an approach has been documented by Duarte and Martí (2007) for MDP. To determine the vertex to be dropped at each iteration, the vertex v with the least weight contribution α_v is chosen. Once the vertex v is removed from the solution, each element of the weight contribution vector α is updated according to the formula $\alpha_i = \alpha_i - w_{iv}$, taking $O(|V|)$ time in total.

3.3. Constrained Neighborhood Local Search

To effectively explore the search space, we apply two neighborhood move operators during the local search phase with the best-improvement strategy. One is the popular $Swap(i, j)$ operator, which exchanges a selected vertex $i \in U$ with a non-selected vertex $j \in V \setminus U$. This operator has proven effective in solving HSP and related problems. The neighborhood size induced by this operator is $|U| \times |V \setminus U|$, since each vertex in U can be exchanged with any vertex in $V \setminus U$. Another operator, $SwapPair(a, b, c, d)$, is also employed to identify additional promising candidate solutions. This operator is triggered only if specific conditions are met (see below). Here, a, b represent the vertices to leave the solution (currently in U), while c, d represent the vertices to join the solution (currently in $V \setminus U$). The neighborhood size induced by this operator is $\binom{|U|}{2} \cdot \binom{|V \setminus U|}{2}$, as it involves selecting two vertices from U and two vertices from $V \setminus U$ for exchange.

These two operators allow extensive exploration of the solution space. However, they generate large neighborhoods. As $|V|$ and $|U|$ grow, exploring the entire neigh-

borhood to identify the best move becomes computationally expensive. To improve the search efficiency, we reduce the two neighborhoods by constraining the vertices involved in *Swap* and *SwapPair* to some small subsets, while ensuring that the best neighboring solutions of the unconstrained (original) neighborhoods are always preserved in the constrained (reduced) neighborhoods. It must be noted that the convenience of applying constrained neighborhoods has been well documented in the heuristic literature, especially in the context of tabu search (Glover and Laguna, 1998) with excellent results. The specific conditions used to obtain the two constrained neighborhoods CN_1 and CN_2 are explained in Sections 3.3.1 and 3.3.2.

As shown in Algorithm 2, for a given solution S, we first examine the constrained neighborhood CN_1 associated to *Swap* operator to reach a local optimum S' (line 1). If the deviation between $f(S')$ and the best objective value $f(S^*)$ is less than the average deviation or negative, the algorithm will search a local optimum by alternatively using the constrained *SwapPair* neighborhood CN_2 and the constrained *Swap* neighborhood CN_1 until no further improvement can be made (lines 5-18).

Finally, to increase the chance of finding the best possible solution, we examine an enlarged search space composed of solutions with $k-1$ or $k+1$ selected vertices. Specifically, in addition to searching in the set of all k-subsets, which defines a k-hyperplane Ω_k, we also search in Ω_{k-1} or Ω_{k+1} as follows. Starting from a solution in Ω_k, we first apply the local search procedure of Algorithm 2 to reach a local optimum solution S containing exactly k vertices. Then, with equal probability, we either greedily remove one vertex from S or add one vertex to S to obtain a solution in either Ω_{k-1} or Ω_{k+1}. Local search is then applied to this new solution. The resulting solution is then repaired to restore a k-subset solution, which is further improved by a final run of the local search procedure.

3.3.1. Swap Operator and Constrained Neighborhood CN_1

Swap(i, j) exchanges a selected vertex $i \in U$ with a non-selected vertex $j \in V \setminus U$. To evaluate a candidate *Swap*(i, j) move for the current solution U, we use the corresponding vertex contributions α_i and α_j, i.e., the total edge weights from i and j, respectively, to the selected vertices in U. The move gain is computed by considering

Algorithm 2 Constrained Neighborhood Local Search $CNLS(AvgD, S)$

Input: An initial solution or a perturbed solution S, the average deviation $AvgD$ between the local optimum and the current best optimum
Output: The local optimal solution S' found

```
/* Use the constrained Swap operator to reach a first local optimum */
S' ← LocalSearch1(S)
D = (f(S*) − f(S'))/f(S*)
if D ≤ AvgD or D < 0 then
    while True do
        /* Use the constrained SwapPair operator to reach a new local optimum */
        S_cur ← S'
        S' ← LocalSearch2(S')
        if f(S') ≤ f(S_cur) then
            break
        end if
        /* Use the constrained Swap operator to reach a new local optimum */
        S_cur ← S'
        S' ← LocalSearch1(S')
        if f(S') ≤ f(S_cur) then
            break
        end if
    end while
end if
return The last local optimal solution S'
```

the two changes induced by the swap. Removing vertex i from U decreases the objective value by α_i, since all edges incident to i within U are deleted. Adding vertex j to the updated set $U \setminus \{i\}$ increases the objective value by $\alpha_j - w_{ij}$. The subtraction of w_{ij} is necessary because α_j is evaluated with respect to the current solution U, where i is still present, and thus includes the edge (i, j), which should not be counted in the move evaluation. Therefore, the change in the objective function $\triangle_{ij}^1$, also called the move gain, is given by

$$\triangle_{ij}^1 = \alpha_j - \alpha_i - w_{ij}. \tag{4}$$

According to Eq. (4), in order to maximize the objective function, we would

replace a vertex $i \in U$ with a small weight contribution α_i by another vertex $j \in V \setminus U$ with a large weight contribution α_j, while considering the edge weight w_{ij}. To improve the search efficiency of our local search procedure, we adopt a neighborhood reduction technique similar to those used in Wu and Hao (2013) and Zhou et al. (2017). Specifically, we identify two distinct sets of vertices X and Y, containing the vertices with a small α and a large α, respectively, as follows:

$$X = \{i \in U \mid \alpha_i \leq MinToOut + \beta\}, \tag{5}$$

$$Y = \{j \in V \setminus U \mid \alpha_j \geq MaxToIn - \beta\}, \tag{6}$$

where $MinToOut = min\{\alpha_i \mid i \in U\}$, $MaxToIn = max\{\alpha_j \mid j \in V \setminus U\}$, and β is related to the weight matrix of the input graph G. The constrained neighborhood CN_1 is defined as $CN_1(S) = \{S' \mid S' = S \setminus \{i\} \cup \{j\}, i \in X, j \in Y\}$.

In general, a smaller value of β results in a more constrained neighborhood. To accelerate the neighborhood examination while preserving the best admissible neighboring solution, we seek a sufficiently small value of β. In Wu and Hao (2013) and Zhou et al. (2017), β is set to $wmax$ and $\frac{\rho}{2}wmax$ ($\rho \geq 1$), respectively, where $wmax$ is the largest edge weight of the input graph G. However, these specific β values do not guarantee that the best neighboring solution is preserved in the constrained neighborhood. Here, we show in Theorem 1 a sufficient lower bound on β able to ensure this property. For convenience, we use $w_{ij} = 0$ for non-existent edges and denote by $wmin$ the minimum edge weight over all vertex pairs.

Theorem 1. *Let $i' \in U$ and $j' \in V \setminus U$ be such that $\alpha_{i'} = MinToOut$ and $\alpha_{j'} = MaxToIn$. A sufficient lower bound on β ensuring that the best Swap move is preserved in the constrained neighborhood is $w_{i'j'} - wmin$.*

Proof. Consider the reference move $Swap(i', j')$, with move gain $\triangle^1_{i'j'} = MaxToIn - MinToOut - w_{i'j'}$. Since the best $Swap$ move must have a move gain no smaller than that of the reference move, it suffices to guarantee that every move discarded by the constrained neighborhood CN_1 has a move gain no greater than $\triangle^1_{i'j'}$. A move is discarded by CN_1 whenever $i \notin X$ or $j \notin Y$. We distinguish three cases.

Case 1: $i \notin X$ and $j \in Y$. Then $\alpha_i > MinToOut + \beta$ and $w_{ij} \geq wmin$. Moreover, the largest possible value of α_j is $MaxToIn$. Thus, the maximal possible move gain in this case satisfies $\triangle_{ij}^1 = \alpha_j - \alpha_i - w_{ij} < MaxToIn - (MinToOut + \beta) - wmin = MaxToIn - MinToOut - \beta - wmin$.

Case 2: $i \in X$ and $j \notin Y$. Then $\alpha_j < MaxToIn - \beta$ and $w_{ij} \geq wmin$. The minimal possible value of α_i is $MinToOut$. Hence, the maximal possible move gain in this case satisfies $\triangle_{ij}^1 = \alpha_j - \alpha_i - w_{ij} < (MaxToIn - \beta) - MinToOut - wmin = MaxToIn - MinToOut - \beta - wmin$.

Case 3: $i \notin X$ and $j \notin Y$. We have $\alpha_i > MinToOut + \beta$, $\alpha_j < MaxToIn - \beta$, and $w_{ij} \geq wmin$. Hence, the maximal possible move gain in this case satisfies $\triangle_{ij}^1 = \alpha_j - \alpha_i - w_{ij} < (MaxToIn - \beta) - (MinToOut + \beta) - wmin = MaxToIn - MinToOut - 2\beta - wmin$.

Together, the above three cases show that any move excluded from CN_1 has a move gain bounded above by $MaxToIn - MinToOut - \beta - wmin$. It is therefore enough to require $MaxToIn - MinToOut - \beta - wmin \leq MaxToIn - MinToOut - w_{i'j'}$, which is equivalent to $\beta \geq w_{i'j'} - wmin$. Under this condition, no move better than the reference move can be discarded, and consequently the best $Swap$ move remains in the constrained neighborhood. □

Accordingly, we set $\beta = w_{i'j'} - wmin$ in our local search procedure to restrict the search to a small neighborhood while ensuring that at least one best $Swap$ move is preserved. Once the swap operation $Swap(i, j)$ is performed, α is updated for each vertex v as $\alpha_v = \alpha_v - w_{vi} + w_{vj}$.

3.3.2. SwapPair Operator and Constrained Neighborhood CN_2

As for the move operator $SwapPair(a, b, c, d)$, let $\eta_{ab} = \alpha_a + \alpha_b - w_{ab}$, $\theta_{cd} = \alpha_c + \alpha_d + w_{cd}$, and $\delta_{abcd} = w_{ac} + w_{ad} + w_{bc} + w_{bd}$, the move gain can be calculated as $\triangle_{abcd}^2 = \theta_{cd} - \eta_{ab} - \delta_{abcd}$. Replacing the vertex pair having a small η value in the solution with an unselected vertex pair having a large θ value should result in a significant improvement in the objective. Based on the way of identifying the constrained neighborhood in Section 3.3.1, we screen these two types of vertex pairs

by the following two sets XP and YP:

$$XP = \{(a, b) \mid a, b \in U, a \neq b, \eta_{ab} \leq MinToOutP + \lambda\}, \tag{7}$$

$$YP = \{(c, d) \mid c, d \in V \setminus U, c \neq d, \theta_{cd} \geq MaxToInP - \lambda\}, \tag{8}$$

where $MinToOutP = min\{\eta_{ab} \mid a, b \in U, a \neq b\}$, $MaxToInP = max\{\theta_{cd} \mid c, d \in V \setminus U, c \neq d\}$, and λ is related to the weight matrix of the input graph. The constrained neighborhood CN_2 is then defined as $CN_2(S) = \{S' \mid S' = S \setminus \{a, b\} \cup \{c, d\}, (a, b) \in XP, (c, d) \in YP\}$. Theorem 2 gives a sufficient lower bound on λ for constructing the constrained neighborhood CN_2.

Theorem 2. *Let (a', b') and (c', d') be such that $\eta_{a'b'} = MinToOutP$ and $\theta_{c'd'} = MaxToInP$. A sufficient lower bound on λ ensuring that the best SwapPair move is preserved in the constrained neighborhood is given by $\delta_{a'b'c'd'} - 4 \times wmin$.*

Proof. Consider the reference move $SwapPair(a', b', c', d')$, with move gain $\triangle^2_{a'b'c'd'} = MaxToInP - MinToOutP - \delta_{a'b'c'd'}$. To ensure that the best $SwapPair$ move is not excluded from CN_2, it is sufficient to show that every discarded move has a move gain no greater than $\triangle^2_{a'b'c'd'}$. A move is discarded whenever $(a, b) \notin XP$ or $(c, d) \notin YP$. Thus, it remains to examine the following three cases.

Case 1: $(a, b) \notin XP$ and $(c, d) \in YP$. In this case, $\eta_{ab} > MinToOutP + \lambda$ and $\delta_{abcd} \geq 4 \times wmin$. Since the largest possible value of θ_{cd} is $MaxToInP$, we obtain $\triangle^2_{abcd} = \theta_{cd} - \eta_{ab} - \delta_{abcd} < MaxToInP - (MinToOutP + \lambda) - 4 \times wmin = MaxToInP - MinToOutP - \lambda - 4 \times wmin$.

Case 2: $(a, b) \in XP$ and $(c, d) \notin YP$. Here, $\theta_{cd} < MaxToInP - \lambda$ and $\delta_{abcd} \geq 4 \times wmin$. On the other hand, the smallest possible value of η_{ab} is $MinToOutP$. It follows that $\triangle^2_{abcd} = \theta_{cd} - \eta_{ab} - \delta_{abcd} < (MaxToInP - \lambda) - MinToOutP - 4 \times wmin = MaxToInP - MinToOutP - \lambda - 4 \times wmin$.

Case 3: $(a, b) \notin XP$ and $(c, d) \notin YP$. In this situation, $\eta_{ab} > MinToOutP + \lambda$, $\theta_{cd} < MaxToInP - \lambda$, and $\delta_{abcd} \geq 4 \times wmin$. Therefore, $\triangle^2_{abcd} = \theta_{cd} - \eta_{ab} - \delta_{abcd} < (MaxToInP - \lambda) - (MinToOutP + \lambda) - 4 \times wmin = MaxToInP - MinToOutP - 2 \times \lambda - 4 \times wmin$.

Thus, in every case, any discarded move has a move gain strictly smaller than $MaxToInP - MinToOutP - \lambda - 4 \times wmin$. To guarantee that no such move can outperform the reference move, it is sufficient to impose $MaxToInP - MinToOutP - \lambda - 4 \times wmin \leq MaxToInP - MinToOutP - \delta_{a'b'c'd'}$, which yields $\lambda \geq \delta_{a'b'c'd'} - 4 \times wmin$. Consequently, when $\lambda \geq \delta_{a'b'c'd'} - 4 \times wmin$, every move better than the reference move is preserved in the constrained neighborhood, and so is the best $SwapPair$ move. □

In our algorithm, λ is set to $\delta_{a'b'c'd'} - 4 \times wmin$ so that the search can be restricted to a small neighborhood, thereby reducing the computation time. In each iteration, the two subsets, XP and YP, can be computed in $O(|U|^2)$ and $O(|V \setminus U|^2)$, respectively. Additionally, identifying the best $SwapPair$ move in CN_2 requires $O(|XP| \times |YP|)$ time. Therefore, the overall time complexity of each iteration is $O(|U|^2 + |V \setminus U|^2 + |XP| \times |YP|)$. Although the unconstrained $SwapPair$ neighborhood can be prohibitively large, this computational burden is substantially mitigated in our setting by the above reduction method. As shown in Section 5.2, this reduction is highly effective on the considered HSP instances. In other problem settings, however, if the screening mechanism was less effective and the reduced neighborhood remained large, the $SwapPair$ operator could still become computationally expensive. In such cases, a more selective use of $SwapPair$ or stricter restrictions on candidate pairs may be needed. Such alternatives mainly aim at reducing the computational burden and may no longer preserve the best move of the whole neighborhood.

3.4. RL-guided Perturbation Selection Strategy

The main idea of RL-BLS perturbation is to introduce an appropriate degree of diversification at different search stages, guiding the algorithm to explore new promising regions. When the search revisits previously encountered local optima, cycles will occur, leading to search stagnation and waste of time. To avoid them, RL-BLS relies on an informed perturbation mechanism that is controlled by two key factors: the type of perturbation T and the jump magnitude L (i.e., the number or the depth of applied perturbation moves). The determination of both factors are based on

the current search stage and historical information on recently visited local optima, stored in a hash table (Benlic and Hao, 2013c). The jump magnitude is directly adjusted based on this information, like in the BLS method, while the perturbation selection is performed through a Q-learning mechanism. In this mechanism, the state is also derived from the same data, capturing features such as cycle patterns. By combining this state representation with the experience accumulated during the search, the algorithm can dynamically select the most suitable perturbation for the given search stage.

3.4.1. Hashing-based Cycle Detection

RL-BLS uses a hash-table mechanism to detect repeated local optima during the search (Benlic and Hao, 2013c). Each time the local search reaches a local optimum S, the algorithm checks whether this solution has been previously encountered. For this purpose, RL-BLS stores up to 500 most recently visited local optima in a hash table.

For a local optimum S, its hash index is computed as $h = \left(\sum_{v \in V} x_v z_v\right) \bmod 100000$, where x_v is a binary indicator equal to 1 if vertex v is selected and 0 otherwise, and z_v is a preset integer value chosen pseudo-randomly from $[1, 131072]$. Each occupied entry of the hash table stores the objective value, the solution vector, and the most recent visit information of the corresponding local optimum.

The algorithm first examines the entry indexed by h to determine whether the newly reached local optimum has been visited before. If this entry is occupied by a different solution, subsequent positions are checked until either an identical solution or an empty entry is found. Once an identical local optimum is found, a cycle is detected and the stored visit information is retrieved and updated. Otherwise, the new local optimum is inserted into the hash table as a newly visited solution.

The hash table stores at most recent 500 local optima so that the memory remains focused on recent search history. Whenever this limit is exceeded, the oldest stored entry is removed.

3.4.2. Perturbation Moves

RL-BLS adopts four types of perturbations: directed, memory-based, random, and large-neighborhood-based (LN-based) perturbations. The first three perturbations are based on the *Swap* operator (see Section 3.3.1) and are executed L times, while the LN-based perturbation uses a destroy-and-reconstruct scheme with a destruction depth set to L.

Directed Perturbation. The directed perturbation takes advantage of the information maintained in a tabu list (Glover and Laguna, 1998) and the constrained neighborhood CN_1 proposed in Section 3.3.1. Since the constrained neighborhood combined with a tabu list that excludes some solutions from consideration may result in an empty neighborhood, we set a larger β value of $\beta = \frac{\rho}{2} \times wmax$ ($\rho = 4$) (Zhou et al., 2017), to allow more neighboring solutions to be considered. This perturbation is more oriented towards search intensification than the other perturbations, since it applies a selection rule that favors the non-forbidden swap moves which cause the least degradation in the objective function. A forbidden (tabu) move involves a vertex i that has been removed from the solution during the last γ iterations (γ is called tabu tenure), or that has been added into the solution in the last $0.7 \times \gamma$ iterations (Zhou et al., 2017). γ takes a random value from a given range related to k (between $0.1 \times k$ and $0.5 \times k$ in our experiment). However, a forbidden swap move is always performed if it leads to a better solution than the best recorded solution (this is called the "aspiration criterion by default" in the tabu search literature).

Memory-based Perturbation. This type of perturbation relies on the memory $EliteSol$, which records the R top local optimal solutions (called elite solutions) encountered during the search. Based on this memory and search history, two indicators $EliteFreq_i$ and $FlipFreq_i$ are calculated, where $EliteFreq_i$ counts how many times vertex i appears in $EliteSol$, and $FlipFreq_i$ counts how often variable x_i changes its value during the search. Each vertex i is assigned a score (Glover et al., 2010) to guide the perturbation:

$$Score_i = \frac{EliteFreq_i(R - EliteFreq_i)}{R^2} + \frac{\tau(MaxFreq - FlipFreq_i)}{MaxFreq}, \qquad (9)$$

where τ is a constant (set to 0.3), $MaxFreq$ is the largest $FlipFreq$ value.

With this scoring function, vertices that appear consistently or never appear in the elite solutions (i.e., the corresponding variable x_i consistently gets the same value of 1 or 0 in the elite solutions) are assigned a lower probability of being selected for perturbation. In addition, vertices that are rarely flipped during the search are given higher probability, helping escape from local optima.

Large-Neighborhood-based Perturbation. The LN-based perturbation introduces diversification by using a destroy-and-reconstruct scheme, instead of the *Swap* operator. Given the jump magnitude L, it modifies the given solution U in two steps. First, a destroy phase removes L vertices from the solution U in a greedy fashion. Vertex removal is based on the vertex's minimum contribution to the solution, specifically those with the smallest α values. This destroy phase yields a partial solution U'. Then, the reconstruct phase adds new vertices to U' from the set $V \setminus U$. These new vertices are chosen based on their maximum contribution (i.e., having the largest α value) until the initial solution size $|U|$ is reached. This approach enables the search to reach more distant regions while remaining guided by quality-based criteria.

Random Perturbation. The random perturbation swaps two vertices chosen uniformly at random from the current solution U and the remaining vertices $V \setminus U$, respectively, without considering move quality or historical information.

3.4.3. Adaptive Mechanism for Jump Magnitude

Each time a local optimum is reached, RL-BLS determines a suitable number of perturbation moves for the upcoming perturbation phase. As shown in Algorithm 3, if a cycle is encountered, the number of perturbation moves is incremented (line 3). If no cycle is detected for a long time, indicating that the search has moved to a distant region, the jump magnitude L is decreased (line 6). Finally, L is limited to take its values between L_{min} and L_{max} (lines 8-12).

The degree of adjustment to the jump magnitude is influenced by the search state. Specifically, if the search repeatedly encounters visited local optimal solutions over consecutive rounds, it indicates that the jump magnitude is insufficient. Inversely, if

the search repeatedly reaches new local optimal solutions, this may be an indication that the jump magnitude is too large and the search is jumping back and forth. To reduce the occurrence of these situations, the number of consecutive search rounds, $ConsR$, is introduced to control the adjustment of the jump magnitude, enabling a more efficient navigation in the search space.

Algorithm 3 Determine the Jump Magnitude $DetermineL(S, L, HT)$

Input: the returned variable of the cycle detection procedure $PreVisit$, the number of search rounds in which new or repeated solutions are encountered consecutively $ConsR$, the iteration number when last cycle was encountered lc, and the current iteration number $Iter$

Output: The jump magnitude L

if $PreVisit \neq -1$ **then**
  $lc = Iter$
  $L = L + 1 + ConsR$
else if $(Iter - lc) > \xi \cdot AvgIter$ **then**
  /* $AvgIter$ is the average iteration number of the two last consecutive cycles and ξ is a constant coefficient */
  $L = L - 1 - ConsR$
end if
if $L > L_{max}$ **then**
  $L = L_{max}$
else if $L < L_{min}$ **then**
  $L = L_{min}$
end if
return the updated jump magnitude L

3.4.4. Q-learning-guided Perturbation Selection

Q-learning is used to dynamically select the most suitable perturbation type during the search. Since the utility of each perturbation varies with the search state, Q-learning employs a Q-value function to estimate the expected long-term reward of performing a specific action (perturbation) at a given state (search stage) (Karimi-Mamaghan et al., 2023; Li et al., 2024). For our purpose of perturbation selection, we specify the states, actions, rewards, Q-table, and transition policy as follows.

State (ST). The state representation focuses on the cyclical behavior of the search and consists of two features: the cycle status st_1 and the cycle-free interval st_2.

Let cl denote the cycle length when the current local optimum has been visited previously, i.e., the number of perturbation rounds between its current occurrence and its most recent previous occurrence. If the current local optimum has not been visited previously, no cycle is detected. The cycle-status feature st_1 is formally defined as

$$st_1 = \begin{cases} 0, & \text{if no cycle is detected in the current round,} \\ 1, & \text{if } 1 \leq cl \leq b_1^{(1)}, \\ 2, & \text{if } b_1^{(1)} < cl \leq b_2^{(1)}, \\ 3, & \text{if } b_2^{(1)} < cl \leq b_3^{(1)}, \\ 4, & \text{if } cl > b_3^{(1)}. \end{cases}$$

Let h denote the number of consecutive perturbation rounds elapsed since the most recently detected cycle. We set $h = 0$ when a cycle is detected in the current round; before the first detected cycle, h is measured from the beginning of the search. The cycle-free interval feature st_2 is defined as

$$st_2 = \begin{cases} 0, & \text{if } 0 \leq h \leq b_1^{(2)}, \\ 1, & \text{if } b_1^{(2)} < h \leq b_2^{(2)}, \\ 2, & \text{if } b_2^{(2)} < h \leq b_3^{(2)}, \\ 3, & \text{if } h > b_3^{(2)}. \end{cases}$$

Therefore, the complete state is represented by $st = (st_1, st_2)$, and the state space contains $5 \times 4 = 20$ possible states.

Action (A). At each state, all types of perturbation are considered as available actions. Let $A = \{0, 1, 2, 3\}$ denote the action set, where the four action indices correspond to the directed, memory-based, LN-based, and random perturbations, respectively. These actions are indexed in increasing order of diversification strength.

Reward ($\mathcal{R}$). The reward mechanism is designed to evaluate the effectiveness of each perturbation action and guide future perturbation selection. Its design follows three principles (Battiti et al., 2008; Benlic and Hao, 2013c; Benlic et al., 2017): 1) prioritize intensification, applying diversification only when necessary; 2) assign higher penalties to larger cycles, which are harder to break; and 3) encourage longer cycle-free search intervals, which indicate more effective exploration.

Specifically, after applying perturbation action a in state st and performing the subsequent local search, a new state $st' = (st'_1, st'_2)$ is obtained. Let $rwd(st, a, st')$ denote the corresponding immediate reward. It is defined as:

$$rwd(st, a, st') = \begin{cases} \phi + (|A| - a) + (st'_2 - 1.5), & \text{if } st'_1 = 0, \\ -(st'_1 + 1) + (st'_2 - 1.5), & \text{otherwise.} \end{cases}$$

Here, ϕ is a strong positive reward assigned when the search reaches a new local optimum without detecting a cycle. The four perturbation actions are ordered in increasing diversification strength, so the bonus term $(|A| - a)$ gives higher rewards to perturbations that introduce less diversification. When a cycle is detected, the penalty term $-(st'_1 + 1)$ ensures that larger cycles receive stronger penalties. In addition, as $st'_2 \in \{0, 1, 2, 3\}$, the term $(st'_2 - 1.5)$ takes values -1.5, -0.5, 0.5, and 1.5, respectively, thereby penalizing frequent cycles and rewarding longer cycle-free intervals. To smooth the potentially noisy outcome of an individual perturbation while gradually discounting outdated information (Zou et al., 2024), the corresponding reward-memory entry is updated as $\mathcal{R}(st, a) = \zeta\mathcal{R}(st, a) + rwd(st, a, st')$, where $\zeta \in [0, 1]$ is the reward-memory decay coefficient, while all other entries of $\mathcal{R}$ remain unchanged.

Q-table $\mathcal{Q}$. Since both the state set and the action set are finite, the Q-table is fully enumerated over all state-action pairs and stores the estimated action value for each pair. After computing $\mathcal{R}(st, a)$, the corresponding Q-value is updated as $\mathcal{Q}(st, a) = (1 - \mu)\mathcal{Q}(st, a) + \mu[\mathcal{R}(st, a) + \kappa \max_{a' \in A} \mathcal{Q}(st', a')]$, where $\mu \in [0, 1]$ is the learning rate and $\kappa \in [0, 1]$ is the discount factor, and the term $\max_{a' \in A} \mathcal{Q}(st', a')$ reflects the maximum expected future reward from st'.

Transition policy. An ϵ-greedy policy is used to guide perturbation selection. Specifically, with probability ϵ, the action a^* that maximizes the Q-value is chosen, while with probability $1-\epsilon$, an action is selected uniformly at random. The parameter ϵ is updated dynamically using the following formula:

$$\epsilon = \epsilon_{base} + \epsilon_{amp} \cdot \tanh\left(\frac{cr - 5000}{2000}\right), \tag{10}$$

where $\epsilon_{base} - \epsilon_{amp}$ and $\epsilon_{base} + \epsilon_{amp}$ are the minimum and maximum probability of greedy selection, and cr is the current search round. The constants 5000 and 2000 control the midpoint and speed of the change, respectively. This setup allows the search to start with a relatively small ϵ, favoring exploration in the early stages, and gradually increase as the search progresses.

4. Computational Results

This section presents computational results and comparative studies with the state-of-the-art methods in the literature.

4.1. Benchmark Instances

To evaluate the efficiency of the RL-BLS algorithm, we perform computational experiments on HSP benchmark instances available in the literature. To complete this test, we also report computational results for the closely related MDP problem, which serves here as a representative special case of HSP.

The 177 HSP instances[1] were introduced by Brimberg et al. (2009), and are grouped into three sets.

- SET I is generated from 27 random graphs with $|V| = 1000$, including 16 sparse graphs with edge densities of 1.0%, 1.2%, ..., 4.0% and 11 graphs with higher edge densities of 20.0%, 22.0%, ..., 40.0%. For each graph, three different k values are considered: 300, 400 and 500, giving 81 instances in total. The edge weights are integers uniformly taken from the interval $[100, 1000]$.

[1] Available upon request.

- SET II includes 16 sparse graphs with $|V| = 3000$ and edge densities ranging from 1.0% to 4.0%. For each graph, k is set to 900, 1200 and 1500, giving a total of 48 problem instances. The edge weights of these instances are integers uniformly taken from the interval $[100, 1000]$.
- SET III is based on 16 sparse graphs and the corresponding k values of the first set (48 instances in total), but the weight of k randomly selected edges is set to a heavy weight (= 10000). Compared to the other two sets, this set of instances is expected to be much harder to solve.

The 80 large MDP instances with at least 2000 elements are taken from the public library MDPLIB 2.0[2], which are grouped into three sets.

- MDG consists of three data sets: MDG-a, MDG-b and MDG-c. Each set contains 20 instances, and all the instances have 100% density with $|V| = 2000$. For MDG-a, $k = 200$ and the edge weights are all integers between 1 and 10 with a uniform distribution. For MDG-b, k is also 200, but the edge weights are real numbers in the interval $(0, 1000]$. In MDG-c, k is set to 300, 400, 500, 600, the edge weights are integers randomly chosen between 1 and 1000.
- b2500 includes 10 instances with $|V| = 2500$, $k = 1000$. This set of instances are all with 10% density, and the edge weights are integers randomly generated from $[-100, 100]$. This dataset contains signed edge weights and is therefore treated as an extended benchmark beyond the basic nonnegative-weight HSP formulation.
- p3000 and p5000 have 5 instances with $|V| = 3000$, $k = 1500$ and 5 instances with $|V| = 5000$, $k = 2500$. The density of these instances is 10%, 30%, 50%, 80%, 100% respectively. The edge weights are integers generated from a $[1, 100]$ uniform distribution.

It is worth noting that the edge weights of the above MDP instances do not verify the triangle inequality property. Except for the signed-weight b2500 set, they can be directly represented as HSP instances. For the b2500 set, although it lies outside this basic formulation, the move-gain formulas and the associated update operations re-

[2]Available at https://www.uv.es/rmarti/paper/mdp.html

main valid for arbitrary real edge weights. Moreover, the proofs of Theorems 1 and 2 do not rely on the nonnegativity of edge weights, but only on the lower bound defined by *wmin*. Therefore, the proposed algorithm and the neighborhood-reduction conditions remain applicable to the b2500 instances.

4.2. Experimental Protocol

The RL-BLS algorithm[3] is implemented in C++, and compiled by GNU gcc 10.2.0 with -O3 flag. The experiments are conducted on a computer equipped with an Intel(R) Gold 6226R(2.90GHz) processor running Linux 3.10.x86_64.

The algorithm uses several parameters, as shown in Table 1. The adopted values for these parameters are listed in the last column, some of which refer to the settings in the reference paper, and some of which are determined by the automatic parameter tuning tool irace (López-Ibáñez et al., 2016). Note that the tuning was conducted in two stages. First, the parameters unrelated to the learning mechanism were tuned for the conventional BLS, i.e., the version without reinforcement learning. Then, after incorporating the learning mechanism, these previously tuned parameters were kept fixed, and only the learning-related parameters were further tuned. This strategy preserves the behavior of the underlying search framework and ensures a fair comparison with the variant without the learning mechanism.

We used 12 randomly selected HSP instances as training instances, and the tuning budget was set to 5000, corresponding to the maximum number of parameter configurations evaluated by irace. These 12 instances include SET I-sparse-1000-6000-300, SET I-sparse-1000-14000-500, SET I-sparse-1000-20000-300, SET I-dense-1000-110000-400, SET I-dense-1000-130000-300, and SET I-dense-1000-180000-500 from SET I; SET II-sparse-3000-15000-1500, SET II-sparse-3000-39000-900, and SET II-sparse-3000-60000-1200 from SET II; and SET III-sparse-1000-7000-500, SET III-sparse-1000-16000-400, and SET III-sparse-1000-19000-500 from SET III. The runtime limit assigned to each instance during tuning is consistent with that used in the main experiments. The resulting parameter configuration was then used across all

[3]Available on https://github.com/hzzheng65-hzz/HSP_RLBLS.git.

experiments, including those on MDP instances, without further parameter tuning.

Table 1: Parameter Settings for RL-BLS

Param.	Description	Range/Ref.	Value
ρ	Neighborhood scale of directed perturbation	[1,10]	4
R	The number of elite solutions	Glover et al. (2010)	8
τ	Flip frequency score coefficient	Glover et al. (2010)	0.3
ξ	Coefficient of average iteration numbers	[0.5,9.5]	1.5
L_{min}	Minimum jump magnitude	$[0.01k,0.2k]$	$0.01k$
L_{max}	Maximum jump magnitude	$[0.2k,k]$	k
$(b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$	Thresholds for cycle status	$\{10, 15, \ldots, 120\}$	$(60, 90, 100)$
$(b_1^{(2)}, b_2^{(2)}, b_3^{(2)})$	Thresholds for cycle-free interval	$\{10, 15, \ldots, 120\}$	$(10, 45, 80)$
ϕ	Positive reward value	{5,10,...20}	10
μ	Learning rate	{0.1, 0.2, 0.3, 0.4, 0.5}	0.2
κ	Discount factor	{0.8, 0.85, 0.9, 0.95}	0.85
ζ	Reward-memory decay coefficient	Zou et al. (2024)	0.95
$\epsilon_{base} - \epsilon_{amp}$	Minimum probability of ϵ-greedy	{0.45, 0.5, 0.55, 0.6, 0.65}	0.5
$\epsilon_{base} + \epsilon_{amp}$	Maximum probability of ϵ-greedy	{0.75, 0.8, 0.85, 0.9, 0.95}	0.85

To obtain the experimental results, we perform 10 runs for each instance using the fixed random seeds $10000 \times \{1, 2, \ldots, 10\}$. Following the literature, the stopping criterion is a cutoff time limit, which is set to a different value for each data set, as shown in Table 2.

Table 2: Main features of HSP benchmark instances (first part), MDP benchmark instances (second part), and the stopping conditions

Benchmark	Dataset	$\#Inst.$	$\|V\|$	k	Density(%)	Edge weights	Weight type	Time limit(s)
HSP	SET I-sparse	48	1000	300, 400, 500	1.0–4.0	$[100, 1000]$	int	200
	SET I-dense	33	1000	300, 400, 500	20.0–40.0	$[100, 1000]$	int	200
	SET II-sparse	48	3000	900, 1200, 1500	1.0–4.0	$[100, 1000]$	int	1000
	SET III-sparse	48	1000	300, 400, 500	1.0–4.0	$[100, 1000] \cup \{10000\}$	int	200
MDP	MDG-a	20	2000	200	100.0	$[1, 10]$	int	600
	MDG-b	20	2000	200	100.0	$(0, 1000]$	real	600
	MDG-c	20	2000	300, 400, 500, 600	100.0	$[1, 1000]$	int	600
	b2500	10	2500	1000	10.0	$[-100, 100]$	int	300
	p3000	5	3000	1500	10.0–100.0	$[1, 100]$	int	600
	p5000	5	5000	2500	10.0–100.0	$[1, 100]$	int	1800

4.3. Comparisons with the State-of-art Algorithms

To evaluate the effectiveness and efficiency of our algorithm, we compare it with the best-performing methods for HSP and MDP. All reference algorithms are implemented in C++ and run under the same experimental conditions as RL-BLS (see

Section 4.2). These include BVNS (Brimberg et al., 2009) and OVNS (Saarinen et al., 2023), that we have re-implemented based on the original papers, and OBMA (Zhou et al., 2017), whose source code is publicly available. In all cases, we adopted the parameter settings in the corresponding original studies and did not perform any additional parameter retuning, to keep the comparison protocol consistent with the original methods. Note that OBMA, originally designed for MDP without considering the triangle inequality, can be directly applied to HSP. RL-BLS, BVNS, and OVNS, designed for HSP, can also be used to solve the aforementioned MDP instances.

4.3.1. Comparative Results for HSP

For the HSP instances, Table 3 summarizes the comparison of RL-BLS with the reference algorithms in terms of the best and average results over 10 runs, and additionally reports the comparison against the best-known solutions (BKS). The BKS used here correspond to the best results reported in Brimberg et al. (2009). The detailed results for each HSP instance are given in Appendix A. For each comparison, the numbers outside the parentheses correspond to the best results, while those between the parentheses concern the average results. The first two columns of Table 3 list the basic information about the instance sets. Column $\#Wins$ denotes the number of instances where RL-BLS outperforms the reference algorithm, $\#Ties$ indicates instances with equal results, and $\#Losses$ shows where RL-BLS is worse. The symbol "-" indicates that the results are unavailable. To assess whether the differences in the results between our algorithm and each reference algorithm are statistically significant across all instances, we conducted a Wilcoxon signed-rank test and report the resulting p-value. In addition, we report the corresponding rank-biserial correlation (RBC) as an effect-size measure. The paired differences are defined as the objective value of the reference algorithm minus that of RL-BLS; a negative RBC indicates an advantage in favor of RL-BLS.

As can be seen from Table 3, RL-BLS demonstrates superior performance to the reference algorithms in solving the HSP instances. In terms of the best results, RL-BLS outperforms BVNS and OVNS in 152 and 150 instances out of 177, respec-

Table 3: Comparison of the best and average results over 10 runs: wins, ties, and losses of RL-BLS versus BVNS, OVNS, OBMA and BKS on the 177 HSP instances

Benchmark	k	RL-BLS vs. BVNS			RL-BLS vs. OVNS			RL-BLS vs. OBMA			RL-BLS vs. BKS		
		#Wins	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*
SET I - sparse	300	15(16)	1(0)	0(0)	16(16)	0(0)	0(0)	2(8)	14(8)	0(0)	9(-)	7(-)	0(-)
	400	13(16)	3(0)	0(0)	16(16)	0(0)	0(0)	2(5)	14(10)	0(1)	10(-)	6(-)	0(-)
	500	14(16)	2(0)	0(0)	15(16)	1(0)	0(0)	1(10)	15(6)	0(0)	12(-)	4(-)	0(-)
SET I - dense	300	6(11)	5(0)	0(0)	3(11)	8(0)	0(0)	0(0)	11(11)	0(0)	2(-)	9(-)	0(-)
	400	4(11)	7(0)	0(0)	2(8)	9(3)	0(0)	0(0)	11(9)	0(2)	0(-)	11(-)	0(-)
	500	4(11)	7(0)	0(0)	2(10)	9(1)	0(0)	0(0)	11(11)	0(0)	0(-)	11(-)	0(-)
SET II - sparse	900	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	15(-)	0(-)	1(-)
	1200	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	14(16)	1(0)	1(0)	16(-)	0(-)	0(-)
	1500	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	15(15)	0(0)	1(1)	16(-)	0(-)	0(-)
SET III - sparse	300	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(-)	0(-)	0(-)
	400	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(-)	0(-)	0(-)
	500	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(16)	0(0)	0(0)	16(-)	0(-)	0(-)
Total		152(177)	25(0)	0(0)	150(173)	27(4)	0(0)	98(118)	77(55)	2(4)	128(-)	48(-)	1(-)
p-value		1.08e-26(8.49e-31)			2.30e-26(3.85e-30)			6.70e-18(9.20e-21)			3.70e-22(-)		
RBC		-1(-1)			-1(-1)			-0.993(-0.975)			-0.982(-)		

tively. Compared with OBMA, RL-BLS obtains 98 wins, 77 ties, and only 2 losses in terms of the best results, and exhibits particularly strong performance on the large SET II instances and the difficult SET III instances. Further, RL-BLS improves 128 BKS values and matches all but one of the remaining instances. When considering the average results, RL-BLS shows even better performances, winning all 177 instances against BVNS and 173 instances against OVNS, without any loss in either comparison. It also records 118 wins, 55 ties, and only 4 losses relative to OBMA. Notably, RL-BLS wins 95 out of 96 SET II and SET III instances in this comparison. The Wilcoxon signed-rank test results further confirm that these differences are statistically significant, as all reported p-values are far below 0.05. Moreover, the corresponding RBC values are -1 for the comparisons with BVNS and OVNS, -0.993 (-0.975) for the comparison with OBMA based on the best (average) results, and -0.982 for the comparison with BKS, indicating extremely strong effect sizes in favor of RL-BLS. Therefore, from the perspective of both the best and average results, we can conclude that RL-BLS is highly competitive for solving HSP.

In Table 4, we report, for each group of instances, the deviation of the best result (f_{best}) and the average result (f_{avg}) of each algorithm over 10 runs from the BKS values, which is calculated as $\%dev_{best} = \frac{f_{best}-BKS}{BKS} \times 100\%$ and $\%dev_{avg} = \frac{f_{avg}-BKS}{BKS} \times 100\%$. Given that HSP is a maximization problem, positive values for $\%dev_{best}$ and $\%dev_{avg}$ indicate an improvement. Note that all values are rounded to

two decimal places. A value of 0 indicates an exact zero deviation, while a value of 0.00 denotes a non-zero deviation that rounds to zero. Dominating values are shown in bold and are based on the actual data rather than the rounded values. For indicative purpose, this table also shows the average computation time (t_{avg}) for each algorithm to reach its best solution over 10 runs.

Table 4: Deviations of the best and average results from the BKS values on the 177 HSP instances over 10 runs

Benchmark	k	BVNS			OVNS			OBMA			RL-BLS		
		$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}
SET I - sparse	300	-0.06	-0.22	84.13	-0.1	-0.21	101.55	0.01	0.01	57.95	**0.02**	**0.02**	**15.13**
	400	-0.04	-0.16	63.84	-0.07	-0.14	113.08	**0.02**	**0.02**	55.06	**0.02**	**0.02**	**11.78**
	500	-0.02	-0.08	37.92	-0.03	-0.07	118.83	**0.01**	**0.01**	60.07	**0.01**	**0.01**	**8.15**
SET I - dense	300	-0.01	-0.05	66.85	-0.01	-0.03	58.37	**0**	**0**	**3.15**	**0**	**0**	15.06
	400	**0**	-0.03	58.88	**0**	-0.01	76.23	**0**	**0**	**2.55**	**0**	**0**	12.08
	500	**0**	-0.01	36.02	**0**	**0**	84.68	**0**	**0**	**3.08**	**0**	**0**	3.29
SET II - sparse	900	-0.11	-0.38	**309.17**	-0.18	-0.32	622.05	0.21	0.16	693.84	**0.27**	**0.26**	575.55
	1200	0	-0.15	**376.38**	-0.14	-0.21	544.73	0.17	0.13	725.17	**0.2**	**0.2**	477.84
	1500	0.03	-0.07	384.27	-0.11	-0.15	497.24	0.12	0.09	724.12	**0.14**	**0.13**	**357.12**
SET III - sparse	300	-3.2	-4.24	99.64	-0.71	-0.92	96.28	-0.57	-0.8	138.44	**0.21**	**0.15**	**27.16**
	400	-1.65	-2.46	85.76	-0.62	-0.77	111.13	-0.12	-0.28	148.54	**0.23**	**0.21**	**62.67**
	500	-1.45	-2.05	**63.31**	-0.54	-0.65	88.41	-0.04	-0.11	149.48	**0.18**	**0.18**	67.6
Average		-0.54	-0.83	138.85	-0.21	-0.29	209.38	-0.02	-0.06	230.12	**0.11**	**0.10**	**136.12**

From this table, we can see that for these HSP instances, RL-BLS finds improved best and average results on SET I-sparse, II, and III, and equal results on SET I-dense. Remarkably, it achieves an overall improvement of 0.11% (the best values) across all three sets. In comparison, OBMA also improves certain instances but struggles on the large SET II and challenging SET III instances, while BVNS and OVNS often fail to reach the BKS, with deviations of -0.54% and -0.21%, respectively. Regarding the average results over 10 runs, RL-BLS attains the best $\%dev_{avg}$ of 0.10%, compared to -0.83% for BVNS, -0.29% for OVNS, and -0.06% for OBMA. Another observation is that RL-BLS often reaches high-quality solutions within reasonable time, and its time-to-best results are competitive on several HSP instance groups. Finally, the small difference between $\%dev_{best}$ and $\%dev_{avg}$ of RL-BLS compared to the reference algorithms suggests that RL-BLS performs more consistently and robustly.

Since 12 of the 177 HSP instances were used for parameter tuning by irace, we further exclude these instances and recompute the aggregate comparisons using the existing results on the remaining 165 non-tuning instances. Table 5 reports the

corresponding results. As shown in the table, RL-BLS retains essentially the same comparative performance after excluding the 12 tuning instances. All Wilcoxon tests remain highly significant, with very strong effect sizes in favor of RL-BLS, indicating that the inclusion of the tuning instances does not materially affect the main comparative conclusions.

Table 5: Aggregate comparison of RL-BLS with the reference algorithms after excluding the 12 instances used for parameter tuning.

	f_{best}					f_{avg}				
	#Wins	*#Ties*	*#Losses*	*p*-value	*RBC*	*#Wins*	*#Ties*	*#Losses*	*p*-value	*RBC*
RL-BLS vs. BVNS	142	23	0	4.74e-25	-1	165	0	0	7.91e-29	-1
RL-BLS vs. OVNS	141	24	0	6.93e-25	-1	161	4	0	3.59e-28	-1
RL-BLS vs. OBMA	91	72	2	9.36e-17	-0.993	109	52	4	2.51e-19	-0.974
RL-BLS vs. BKS	121	44	0	1.35e-21	-1	–	–	–	–	–

4.3.2. Comparative Results for MDP

To evaluate the robustness and transferability of RL-BLS on a closely related problem, we test it on the 80 benchmark instances of MDP, which, as mentioned above, is a special case of HSP. The comparison results are provided in Tables 6 and 7, showing the same information as in Tables 3 and 4. In this case, the BKS used for comparison correspond to the best results previously reported in Zhou et al. (2017).

Table 6: Comparison of the best and average results over 10 runs: wins, ties, and losses of RL-BLS versus BVNS, OVNS, OBMA and BKS on the 80 MDP instances

Benchmark	*k*	RL-BLS vs. BVNS			RL-BLS vs. OVNS			RL-BLS vs. OBMA			RL-BLS vs. BKS		
		#Wins	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*	*#Wins*	*#Ties*	*#Losses*
MDG-a	200	19 (20)	1 (0)	0 (0)	15 (20)	5 (0)	0 (0)	0 (0)	20 (20)	0 (0)	0 (-)	20 (-)	0 (-)
MDG-b	200	18 (20)	2 (0)	0 (0)	19 (20)	1 (0)	0 (0)	0 (1)	20 (15)	0 (4)	0 (-)	19 (-)	1 (-)
MDG-c	300-600	20 (20)	0 (0)	0 (0)	20 (20)	0 (0)	0 (0)	0 (2)	17 (1)	3 (17)	0 (-)	10 (-)	10 (-)
b2500	1000	10 (10)	0 (0)	0 (0)	10 (10)	0 (0)	0 (0)	0 (0)	7 (0)	3 (10)	0 (-)	7 (-)	3 (-)
p3000	1500	5 (5)	0 (0)	0 (0)	5 (5)	0 (0)	0 (0)	0 (0)	5 (1)	0 (4)	0 (-)	5 (-)	0 (-)
p5000	2500	5 (5)	0 (0)	0 (0)	5 (5)	0 (0)	0 (0)	1 (0)	1 (0)	3 (5)	0 (-)	1 (-)	4 (-)
Total		77 (80)	3 (0)	0 (0)	74 (80)	6 (0)	0 (0)	1 (3)	70 (37)	9 (40)	0 (-)	62 (-)	18 (-)
p-value		2.46e-14(7.85e-15)			7.73e-14(7.85e-15)			1.25e-2(9.97e-10)			1.96e-04(-)		
RBC		-1(-1)			-0.990(-1)			0.891(0.930)			1(-)		

As shown in Table 6, RL-BLS outperforms, for the best results, BVNS and OVNS by a large margin, with better results in 77 and 74 out of 80 instances, respectively. In terms of the average results, RL-BLS further extends its dominance by winning

Table 7: Deviations of the best and average results from the BKS values on the 80 MDP instances over 10 runs

Benchmark	k	BVNS			OVNS			OBMA			RL-BLS		
		$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}	$\%dev_{best}$	$\%dev_{avg}$	t_{avg}
MDG-a	200	-0.05	-0.18	231.3	-0.03	-0.11	209.66	**0**	**0**	**41.33**	**0**	**0**	108.4
MDG-b	200	-0.06	-0.16	**224.17**	-0.06	-0.19	584.43	**0**	**0**	286.5	0.00	0.00	362.81
MDG-c	300-600	-0.09	-0.16	236.86	-0.06	-0.1	300.16	**-0.02**	**-0.02**	218.18	**-0.02**	-0.03	**201.01**
b2500	1000	-0.3	-0.6	163.59	-0.26	-0.42	187.08	**0**	**0**	**100.23**	0.00	-0.01	116.02
p3000	1500	-0.01	-0.03	333.25	-0.01	-0.02	375.29	**0**	**0**	**127.29**	**0**	**0**	208.43
p5000	2500	-0.02	-0.03	1244.54	-0.01	-0.01	996.38	**0**	**0**	**719.75**	0.00	0.00	793.45
Average		-0.09	-0.19	405.62	-0.07	-0.14	442.17	**0.00**	**0.00**	248.88	0.00	-0.01	298.35

all instances against BVNS and OVNS. Compared with OBMA, RL-BLS achieves 1 win, 70 ties and 9 losses on the best results and 3 wins, 37 ties and 40 losses on the average results. This indicates that while RL-BLS is very effective against the single trajectory algorithms BVNS and OVNS, the population-based OBMA algorithm, which is dedicated to MDP, remains a strong competitor for the problem. This overall observation is consistent with the statistical results. Table 7 shows that both RL-BLS and OBMA achieve almost zero deviation from the BKS for the best results. For the average results, RL-BLS performs slightly worse than OBMA, with a deviation of -0.01%, which is worse than OBMA's deviation (0.00%), but better than BVNS (-0.19%) and OVNS (-0.14%).

We also tested several other widely used MDP instances that were not included in the main benchmark set. RL-BLS reached the BKS on all of them in a very short time. As these instances are much easier and do not affect the main conclusions, we do not include them here.

4.4. Additional Computational Results on Related Problems

To further examine the applicability of RL-BLS beyond the main HSP and MDP problems, we also tested the algorithm on three related problems: Densest k-Subgraph (DkS) (Kortsarz and Peleg, 1993), Maximum Clique (Wu and Hao, 2015) and Hidden Clique (Schauer, 2016). The detailed results are provided in our GitHub repository: `https://github.com/hzzheng65-hzz/HSP_RLBLS.git`.

For the DkS problem, we used 180 standard benchmark instances from the lit-

erature[4]. Our experiments show that these instances are easy for our algorithm: RL-BLS reaches the known optimum for all tested instances within a very short time, with an average runtime below 0.1 seconds.

For the clique-recovery experiment based on Maximum Clique benchmarks, we considered 72 challenging instances from the DIMACS and BHOSLIB benchmark sets[5]. To embed this setting into the HSP framework, we assigned unit weights to all edges and used a target clique size as the input cardinality parameter k. For each instance, we considered four target values, namely k^*, $k^* - 1$, $k^* - 2$, and $k^* - 3$, where k^* denotes the best-known clique size reported in the benchmark library. We then tested whether RL-BLS could recover a clique of the corresponding target size through the associated HSP formulation. Under a cutoff time of 2000 seconds, RL-BLS successfully recovered a clique of the target size on 60, 66, 68, and 71 out of the 72 instances for $k = k^*$, $k = k^* - 1$, $k = k^* - 2$, and $k = k^* - 3$, respectively. These results indicate that RL-BLS is able to recover maximum cliques on a large majority of the tested instances and, on the remaining most challenging cases, usually finds cliques whose sizes are very close to the best-known clique sizes.

For the hidden-clique recovery experiment, we generated 40 instances according to the instance generation procedure described in Schauer (2016). On all these instances, RL-BLS successfully recovered the hidden clique, with an average runtime below 0.2 seconds. This shows that the proposed framework can also be effectively applied to this structured related setting.

5. Ablation Analysis and Additional Comparative Studies

5.1. Ablation Analysis of Key Components

We evaluate the contributions of three key components of RL-BLS: the *SwapPair* operator, the RL-guided perturbation selection mechanism, and the LN-based perturbation. To this end, we create three algorithm variants and compare them with RL-BLS. In RL-BLS$^{\text{SP-}}$, we disable the *SwapPair* operator. In BLS, we replace the

[4]Available at https://cedric.cnam.fr/~lamberta/Library/k-cluster.html.
[5]Available at https://iridia.ulb.ac.be/~fmascia/maximum_clique/.

RL-guided perturbation selection with the conventional perturbation selection (Benlic and Hao, 2013c). In RL-BLS$^{\text{LN-}}$, we disable the LN-based perturbation while keeping the other perturbation types. All variants keep the other algorithmic components of RL-BLS unchanged.

We run these algorithms on the 177 HSP instances using the same experimental setup as RL-BLS. Table 8 presents the comparative results of RL-BLS against these three variants in terms of the best (f_{best}) and average (f_{avg}) results over 10 runs. The results show that removing the *SwapPair* operator (RL-BLS$^{\text{SP-}}$) leads to a clear performance deterioration: RL-BLS achieves 92 wins in f_{best} and 100 wins in f_{avg}, with *RBC* values close to -1. This demonstrates that *SwapPair* is a key operator that contributes substantially to both solution quality and stability. Compare to the conventional BLS, our RL-BLS algorithm equipped with the learning-guided perturbation selection, achieves 43 wins in f_{best} and 66 wins in f_{avg}. This highlights the effectiveness of RL guidance in choosing perturbations, resulting in better overall performance compared to the conventional strategy. Finally, removing the LN-based perturbation also degrades performance noticeably: RL-BLS obtains 39 wins in f_{best} and 61 wins in f_{avg} against RL-BLS$^{\text{LN-}}$. This shows that the LN-based perturbation provides an additional and meaningful contribution to search diversification. Overall, all three components contribute positively to the performance of RL-BLS.

Table 8: Comparison of the best and average results over 10 runs of RL-BLS against three variants.

	f_{best}					f_{avg}				
	#Wins	*#Ties*	*#Losses*	*p*-value	*RBC*	*#Wins*	*#Ties*	*#Losses*	*p*-value	*RBC*
RL-BLS vs. RL-BLS$^{\text{SP-}}$	92	84	1	3.95e-16	-0.972	100	76	1	3.10e-18	-0.998
RL-BLS vs. BLS	43	116	18	2.40e-03	-0.447	66	81	30	7.46e-04	-0.408
RL-BLS vs. RL-BLS$^{\text{LN-}}$	39	120	18	6.20e-03	-0.417	61	82	34	1.20e-03	-0.382

For a dynamic view of the contribution of these components, we also compare the convergence behavior of RL-BLS and its three variants on four representative HSP instances, as shown in Figure 1. The x-axis represents the running time (in seconds), while the y-axis reports the best objective value found, averaged over 10 runs. Figure 1 provides additional insight into how the three components affect the search process. On the representative sparse instance from SET I, all methods eventually

reach very similar final values, with RL-BLS reaching a high-quality solution earlier, while BLS and RL-BLS$^{\text{LN-}}$ improve more gradually and RL-BLS$^{\text{SP-}}$ terminates at a slightly lower objective value. On the dense SET I instance, the four methods converge to almost indistinguishable final values, although RL-BLS reaches the common final objective value somewhat earlier, indicating that this comparatively easier case leaves limited room for the different perturbation mechanisms to manifest clear advantages. By contrast, on the more challenging SET II and SET III instances, the benefit of the full RL-BLS framework becomes much more evident. In particular, removing the *SwapPair* operator leads to the largest deterioration, with RL-BLS$^{\text{SP-}}$ remaining clearly below the other methods throughout most of the run. Disabling the LN-based perturbation also weakens performance, although the degradation is less severe than that caused by removing *SwapPair*. Compared with the conventional BLS, RL-BLS generally reaches high-quality solutions earlier and attains better final values on these harder instances. These observations are consistent with the ablation results in Table 8 and further confirm that *SwapPair* is the most critical component, while the RL-guided perturbation selection and the LN-based perturbation provide complementary improvements in search guidance and diversification.

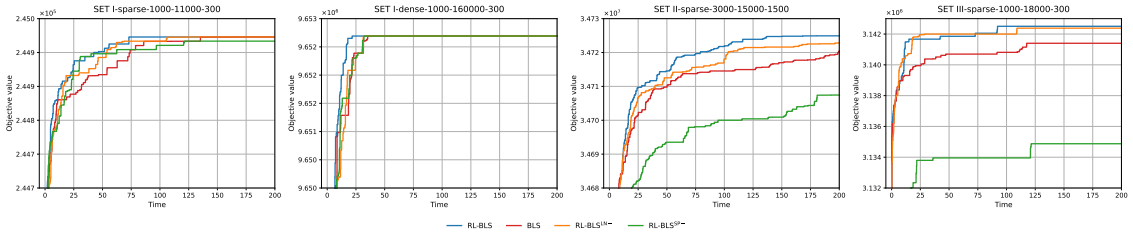

Figure 1: Effect of main components on the convergence of RL-BLS

5.2. Effectiveness of the Neighborhood Reduction Strategy

We evaluate the computational gain of the constrained neighborhoods CN_1 (*Swap*) and CN_2 (*SwapPair*). Note that all of the considered neighborhoods preserve the best possible move. With the best-improvement strategy for neighborhood examination, the same best move is selected in each iteration across all studied neighborhoods. Therefore, rather than comparing solution quality, we focus on neighborhood

size, which directly determines the computational effort per iteration and thus the practical scalability of the local search phase. For the *Swap* neighborhood, we study the effect of the β parameter that controls the neighborhood size, and compare the neighborhood using our chosen β value given by Theorem 1 and two alternative values $\beta = wmax$ (Wu and Hao, 2013), and $\beta = 2wmax$ (Zhou et al., 2017). For the *SwapPair* neighborhood, we compare the constrained neighborhood and the unconstrained neighborhood. For all the cases studied, the size of each neighborhood was recorded at each iteration. Specifically, 5,000 iterations were performed for the *Swap* neighborhood and 1,000 iterations for the *SwapPair* neighborhood. Figures 2 and 4 show the average neighborhood sizes (on a logarithmic scale) over the predefined iterations as solid lines and the ratio of iterations where our constrained neighborhoods contain only a single solution as dashed lines. The horizontal axis represents instances, sorted in non-decreasing order of $|E|$ and k.

Figure 2 shows that for the *Swap* neighborhood, our choice of β significantly reduces the average number of neighboring solutions explored per iteration in all instances, compared to the other two options. For example, in SET II-sparse-3000-15000-900, the average size of our constrained neighborhood is 947, against 82,543 with $\beta = wmax$, and 726,283 with $\beta = 2wmax$, corresponding to neighborhood magnitudes of 2.97, 4.92, and 5.86, respectively. The dashed line depicts the ratio of iterations where CN_1 contains only one solution, reaching up to 82%, 65%, 92%, and 83% for the four instance sets. Such reductions imply that the best move can often be identified directly with $O(|V|)$ effort, greatly enhancing local search efficiency.

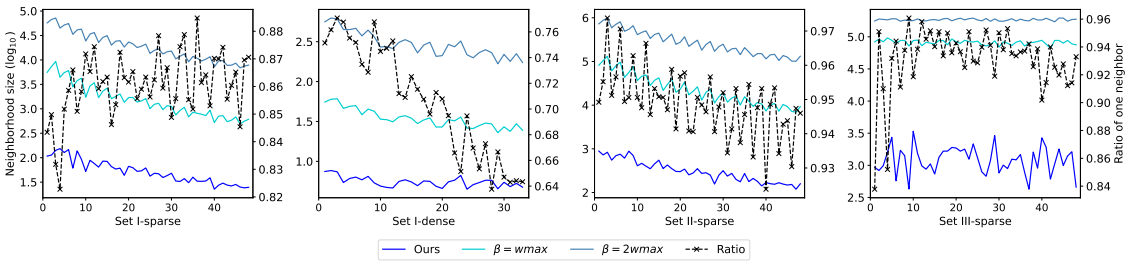

Figure 2: Size comparison of different constrained neighborhoods induced by the *Swap* operator

To complement this analysis, Figure 3 shows the distribution of the percentages of

the three constrained neighborhood sizes relative to the unconstrained neighborhood size over all iterations for four instances. Our choice of β consistently produces smaller percentages, indicating that our constrained neighborhood is much smaller, allowing the best move to be found more quickly at each iteration of the algorithm.

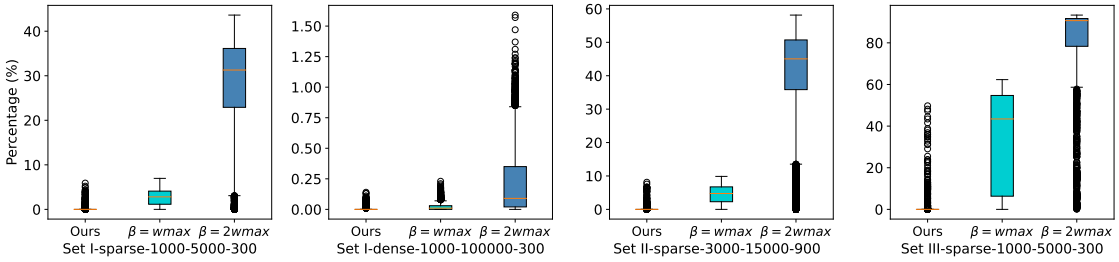

Figure 3: The percentage of the three constrained neighborhoods relative to the unconstrained neighborhood for varying β values

Similarly, Figure 4 demonstrates that, for the *SwapPair* neighborhood, we obtain a reduction of the neighborhood by a factor of 10^3 to 10^8. Moreover, the dashed lines lie consistently at a high level, implying that in most of the neighborhood explorations the best move can be found once the reference vertex pairs are determined. These empirical results also explain why *SwapPair* does not become a bottleneck on the considered HSP instances.

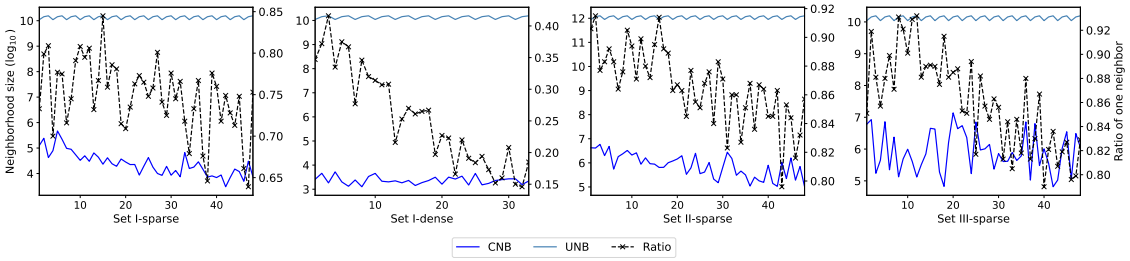

Figure 4: Size comparison of the constrained and unconstrained neighborhoods induced by the *SwapPair* operator

5.3. Analysis of the Adaptive Jump Magnitude Mechanism

To show the effectiveness of the adaptive jump magnitude mechanism of the RL-BLS algorithm, we replace the adaptive mechanism by setting the jump magnitude

to different fixed values $0.01k$, $0.05k$, $0.1k$, $0.2k$, and $0.3k$. We run RL-BLS with its default adaptive jump magnitude and with these fixed magnitudes, using the same stopping criteria as shown in Table 2. For this study and for illustrative purposes, we selected four representative instances from each type of HSP dataset: SET I-sparse-1000-5000-300, SET I-dense-1000-100000-300, SET II-sparse-3000-39000-1200, and SET III-sparse-1000-12000-400.

Figure 5 shows the convergence curves of the algorithm with its default adaptive jump magnitude against the fixed jump magnitudes, where the x-axis represents the running time (in seconds) and the y-axis denotes the best objective value found (averaged over 10 runs). From this figure, we can observe that the adaptive jump magnitude performs consistently well across instances of different scales. In contrast, using the same fixed jump magnitude for instances of varying sizes significantly limits the algorithm performance, as an inappropriate jump magnitude can cause either insufficient or excessive diversification.

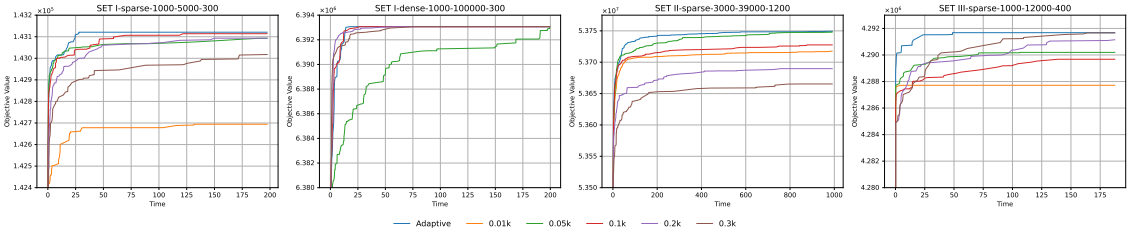

Figure 5: Effect of fixed and adaptive jump magnitudes on the convergence of RL-BLS

6. Conclusion

This paper presents RL-BLS, a Reinforcement Learning-guided Breakout Local Search algorithm for the Heaviest k-Subgraph Problem. The algorithm enhances the traditional BLS framework by a Q-learning-guided perturbation selection mechanism that enables it to dynamically select the most promising perturbation at different search stages. In addition, RL-BLS incorporates two effective neighborhood structures based on the *Swap* and *SwapPair* operators combined with a neighborhood

reduction strategy that substantially reduces the neighborhood size and improves the practical scalability of the local search phase, especially on large instances.

Computational results on 177 HSP and 80 MDP previously reported instances demonstrate the effectiveness and competitiveness of RL-BLS. For HSP, it achieves the best-known solutions for all but one instance, and provides improved best results (new lower bounds) for 128 instances. For MDP, RL-BLS achieves the best-known solutions in most of cases. Additionally, RL-BLS also shows strong performance on the Densest k-Subgraph, Maximum Clique, and Hidden Clique problems. Future research could focus on extending and validating the adaptability of the RL-BLS framework to other graph-optimization problems, as well as exploring advanced reinforcement-learning techniques such as deep learning and multi-agent learning. Another area of focus would be hybridizing RL-BLS with population-based hybrid methods to further improve performance.

Acknowledgement

We are grateful to the editors and reviewers for their comments, which have helped us to improve the work significantly. This research has been partially supported by the National Natural Science Foundation of China (Grant No. 72371200). This work is part of the project PID2024-160226OB-C21, funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU in a program of the Spanish government. It has also been supported by the Generalitat Valenciana (CIAICO/2026/100). Support from the China Scholarship Council (CSC No. 202306290083) is also acknowledged.

References

Angioni, D., Archetti, C., Speranza, M.G., 2025. Neural combinatorial optimization: A tutorial. Computers & Operations Research 182, 107102.

Battiti, R., Brunato, M., Mascia, F., 2008. Reactive search and intelligent optimization. volume 45. Springer Science & Business Media.

Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research 290, 405–421.

Benlic, U., Epitropakis, M.G., Burke, E.K., 2017. A hybrid breakout local search and reinforcement learning approach to the vertex separator problem. European Journal of Operational Research 261, 803–818.

Benlic, U., Hao, J.K., 2013a. Breakout local search for maximum clique problems. Computers & Operations Research 40, 192–206.

Benlic, U., Hao, J.K., 2013b. Breakout local search for the quadratic assignment problem. Applied Mathematics and Computation 219, 4800–4815.

Benlic, U., Hao, J.K., 2013c. Breakout local search for the vertex separator problem, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, AAAI Press. pp. 461–467.

Billionnet, A., 2005. Different formulations for solving the heaviest k-subgraph problem. INFOR: Information Systems and Operational Research 43, 171–186.

Brimberg, J., Mladenović, N., Urošević, D., Ngai, E., 2009. Variable neighborhood search for the heaviest k-subgraph. Computers & Operations Research 36, 2885–2891.

Corneil, D.G., Perl, Y., 1984. Clustering and domination in perfect graphs. Discrete Applied Mathematics 9, 27–39.

De Freitas, A.R.R., Guimarães, F.G., Pedrosa Silva, R.C., Souza, M.J.F., 2014. Memetic self-adaptive evolution strategies applied to the maximum diversity problem. Optimization Letters 8, 705–714.

Duarte, A., Martí, R., 2007. Tabu search and GRASP for the maximum diversity problem. European Journal of Operational Research 178, 71–84.

El Krari, M., Ahiod, B., El Benani, B., 2020. A memetic algorithm based on breakout local search for the generalized traveling salesman problem. Applied Artificial Intelligence 34, 537–549.

Gajewar, A., Das Sarma, A., 2012. Multi-skill collaborative teams based on densest subgraphs, in: Proceedings of the 2012 SIAM International Conference on Data Mining, SIAM. pp. 165–176.

Gallego, M., Duarte, A., Laguna, M., Martí, R., 2009. Hybrid heuristics for the maximum diversity problem. Computational Optimization and Applications 44, 411–426.

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

Gibson, D., Kumar, R., Tomkins, A., 2005. Discovering large dense subgraphs in massive graphs, in: Proceedings of the 31st International Conference on Very large Data Bases, pp. 721–732.

Glover, F., Laguna, M., 1998. Tabu search. Springer.

Glover, F., Lü, Z., Hao, J.K., 2010. Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR 8, 239–253.

He, M., Wu, Q., Lu, Y., 2022. Breakout local search for the cyclic cutwidth minimization problem. Journal of Heuristics 28, 583–618.

Karimi-Mamaghan, M., Mohammadi, M., Pasdeloup, B., Meyer, P., 2023. Learning to select operators in meta-heuristics: An integration of q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 304, 1296–1330.

Kortsarz, G., Peleg, D., 1993. On choosing a dense subgraph, in: Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science, pp. 692–701.

Krislock, N., Malick, J., Roupin, F., 2016. Computational results of a semidefinite branch-and-bound algorithm for k-cluster. Computers & Operations Research 66, 153–159.

Kuby, M.J., 1988. Programming models for facility dispersion: the p-dispersion and maxisum dispersion problems. Mathematical and Computer Modelling 10, 792.

Lai, X., Hao, J., Yue, D., Gao, H., 2018. Diversification-driven memetic algorithm for the maximum diversity problem, in: Proceedings of 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems, IEEE. pp. 310–314.

Letsios, M., Balalau, O.D., Danisch, M., Orsini, E., Sozio, M., 2016. Finding heaviest k-subgraphs and events in social media, in: Proceedings of 2016 IEEE 16th International Conference on Data Mining Workshops, IEEE. pp. 113–120.

Li, M., Hao, J.K., Wu, Q., 2024. A flow based formulation and a reinforcement learning based strategic oscillation for cross-dock door assignment. European Journal of Operational Research 312, 473–492.

Liu, X., Chen, J., Wang, M., Wang, Y., Su, Z., Lü, Z., 2022. A two-phase tabu search based evolutionary algorithm for the maximum diversity problem. Discrete Optimization 44, 100613.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

Macambira, E.M., 2002. An application of tabu search heuristic for the maximum edge-weighted subgraph problem. Annals of Operations Research 117, 175–190.

Martí, R., Martínez-Gavara, A., 2023. Discrete diversity and dispersion maximization. A tutorial on metaheuristic optimization. Springer Optimization and Its Applications 204, Springer Switzerland AG.

Martí, R., Martínez-Gavara, A., Pérez-Peló, S., Sánchez-Oro, J., 2022. A review on discrete diversity and dispersion maximization from an OR perspective. European Journal of Operational Research 299, 795–813.

Palubeckis, G., 2007. Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation 189, 371–383.

Prokopyev, O.A., Kong, N., Martínez-Torres, D.L., 2009. The equitable dispersion problem. European Journal of Operational Research 197, 59–67.

Saarinen, V.P., Chen, T.H.Y., Kivelä, M., 2023. OVNS: Opportunistic variable neighborhood search for heaviest subgraph problem in social networks. URL: https://arxiv.org/abs/2305.19729, arXiv:2305.19729.

Schauer, J., 2016. Asymptotic behavior of the quadratic knapsack problem. European Journal of Operational Research 255, 357–363.

Singh, H., Kumar, M., Aggarwal, P., 2019. Approximation of heaviest k-subgraph problem by size reduction of input graph, in: Proceedings of 2nd International Conference on Communication, Computing and Networking, NITTTR Chandigarh, India, Springer. pp. 599–605.

Sotirov, R., 2020. On solving the densest k-subgraph problem on large graphs. Optimization Methods and Software 35, 1160–1178.

Sutton, R., Barto, A., 1998. Reinforcement learning: An introduction. IEEE Transactions on Neural Networks 9, 1054–1054.

Wang, Y., Hao, J.K., Glover, F., Lü, Z., 2014. A tabu search based memetic algorithm for the maximum diversity problem. Engineering Applications of Artificial Intelligence 27, 103–114.

Watkins, C.J., Dayan, P., 1992. Q-learning. Machine Learning 8, 279–292.

Wu, Q., Hao, J.K., 2013. A hybrid metaheuristic method for the maximum diversity problem. European Journal of Operational Research 231, 452–464.

Wu, Q., Hao, J.K., 2015. A review on algorithms for maximum clique problems. European Journal of Operational Research 242, 693–709.

Zhou, Y., Hao, J.K., Duval, B., 2017. Opposition-based memetic search for the maximum diversity problem. IEEE Transactions on Evolutionary Computation 21, 731–745.

Zou, Y., Hao, J.K., Wu, Q., 2024. A reinforcement learning guided hybrid evolutionary algorithm for the latency location routing problem. Computers & Operations Research 170, 106758.

Appendix A. Detailed results of the 177 HSP instances

We show in Tables A.9–A.12 the detailed comparative results obtained by BVNS (Brimberg et al., 2009), OVNS (Saarinen et al., 2023), OBMA (Zhou et al., 2017), and RL-BLS (this work) on the four sets of 177 HSP instances. The results were obtained by running the codes of these algorithms on the same computer, following the experimental protocol of Section 4.2.

The first two columns give the information about the instances, including the number of vertices to select k, and the number of the edges $|E|$ (the number of vertices $|V|$ is given in the title of each table). Column Updated BKS reports the updated best-known value for each instance after this study. It is obtained as the best value among the previous BKS reported in Brimberg et al. (2009) and the best results of all compared algorithms including our RL-BLS algorithm. The remaining columns present, for each algorithm and instance, the best result (f_{best}) and the average result (f_{avg}) from the 10 independent runs. Additionally, the average running time per run needed to reach its best result (t_{avg}) is also given. The dominating values are shown in bold.

Table A.9: Detailed results of each instance in SET I-sparse ($|V| = 1000$) obtained by RL-BLS (this work) and the reference algorithms BVNS, OVNS, OBMA.

k	$\|E\|$	Updated BKS	BVNS			OVNS			OBMA			RL-BLS		
			f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
300	5000	143121	**143121**	142589.2	110.7	142644	142294.0	129.8	143050	143011.0	106.9	**143121**	**143121.0**	18.1
	6000	162237	161890	161732.5	102.9	161718	161526.7	118.7	162198	162196.1	133.3	**162237**	**162237.0**	4.3
	7000	179576	179504	179201.1	95.4	179415	179216.7	103.0	**179576**	179572.7	78.6	**179576**	**179576.0**	32.4
	8000	196410	196205	195817.2	71.3	196109	195748.0	129.5	**196410**	196404.3	81.4	**196410**	**196410.0**	8.3
	9000	213132	213112	212739.0	98.5	212504	212082.8	133.8	**213132**	213130.0	38.8	**213132**	**213132.0**	19.3
	10000	228563	228529	228209.7	80.7	228494	228133.1	83.9	**228563**	228560.0	90.6	**228563**	**228563.0**	19.9
	11000	244923	244375	244091.5	93.7	244698	244254.1	110.6	**244923**	244891.8	102.4	**244923**	**244923.0**	78.8
	12000	261229	260764	260441.6	87.9	261086	260601.7	90.8	**261229**	**261229.0**	27.5	**261229**	**261229.0**	4.5
	13000	277554	277415	276686.4	82.4	277383	277250.5	114.1	**277554**	**277554.0**	49.8	**277554**	**277554.0**	14.5
	14000	293675	293531	292907.5	66.8	293504	293333.6	94.8	**293675**	**293675.0**	31.5	**293675**	**293675.0**	13.2
	15000	309193	309174	308509.3	84.5	309074	308694.8	112.2	**309193**	**309193.0**	35.8	**309193**	**309193.0**	6.6
	16000	324251	323993	323409.8	95.7	323917	323559.7	85.2	**324251**	**324251.0**	18.3	**324251**	**324251.0**	1.0
	17000	338496	338269	337955.4	68.8	338261	338147.6	75.2	**338496**	**338496.0**	16.9	**338496**	**338496.0**	2.9
	18000	354086	354071	353818.5	48.8	353925	353915.1	51.1	**354086**	**354086.0**	11.6	**354086**	**354086.0**	1.9
	19000	368312	367856	367520.0	83.4	367825	367704.5	92.2	**368312**	**368312.0**	27.0	**368312**	**368312.0**	15.4
	20000	384053	384009	383799.3	74.3	384034	383931.1	99.9	**384053**	384052.6	76.9	**384053**	**384053.0**	1.0
400	5000	206769	206634	206254.2	87.6	206559	206357.3	113.7	206735	206706.3	121.1	**206769**	**206769.0**	5.6
	6000	236626	236334	236193.3	71.1	236077	235843.1	135.8	**236626**	**236613.1**	109.3	**236626**	236600.2	74.4
	7000	264902	264751	264327.9	77.3	264596	264489.8	114.0	**264902**	264896.9	102.9	**264902**	**264902.0**	2.8
	8000	292979	292775	292538.4	67.3	292593	292228.0	91.1	**292979**	292978.7	60.9	**292979**	**292979.0**	2.3
	9000	318883	318860	318164.0	89.4	318198	318024.1	110.6	**318883**	318880.6	102.9	**318883**	**318883.0**	10.6
	10000	344347	343893	343559.9	118.8	344289	343877.5	119.2	344339	344327.9	114.7	**344347**	**344347.0**	13.8
	11000	371297	370694	370038.0	26.3	370918	370475.5	146.8	**371297**	**371297.0**	37.8	**371297**	**371297.0**	4.8
	12000	397533	397467	396299.9	29.1	397416	397113.9	77.7	**397533**	**397533.0**	58.8	**397533**	**397533.0**	2.2
	13000	423966	423420	423174.0	90.2	423423	423333.2	143.8	**423966**	**423966.0**	18.1	**423966**	**423966.0**	10.0
	14000	449570	449523	448721.0	82.4	449111	448788.9	100.0	**449570**	**449570.0**	7.5	**449570**	**449570.0**	2.1
	15000	474588	**474588**	474038.9	24.4	474320	473907.4	117.0	**474588**	**474588.0**	57.2	**474588**	**474588.0**	12.8
	16000	500392	**500392**	499872.0	72.1	500218	499958.7	112.1	**500392**	**500392.0**	14.4	**500392**	**500392.0**	1.3
	17000	524528	524045	523864.1	51.2	524294	523975.5	99.2	**524528**	**524528.0**	20.0	**524528**	**524528.0**	5.4
	18000	549537	549163	548887.3	23.9	549264	549086.6	103.6	**549537**	**549537.0**	19.2	**549537**	**549537.0**	2.7
	19000	573742	**573742**	573480.9	60.4	573685	573415.5	130.2	**573742**	**573742.0**	21.2	**573742**	**573742.0**	7.3
	20000	599027	598850	598581.9	50.0	598635	598492.6	94.6	**599027**	**599027.0**	14.7	**599027**	**599027.0**	30.1
500	5000	275207	274950	274743.7	81.6	274934	274792.0	111.0	**275207**	275181.6	103.7	**275207**	**275207.0**	12.3
	6000	317846	317520	317116.4	11.4	317464	317215.4	95.7	317830	317829.2	80.5	**317846**	**317846.0**	12.0
	7000	359654	359508	359287.3	24.8	359494	359262.4	139.4	**359654**	359643.2	63.0	**359654**	**359654.0**	3.3
	8000	400073	399865	399585.3	24.8	399888	399645.2	125.9	**400073**	**400073.0**	27.1	**400073**	**400073.0**	1.7
	9000	438484	438405	438306.8	77.0	438294	438168.2	122.9	**438484**	**438484.0**	56.9	**438484**	**438484.0**	1.6
	10000	476317	476296	475758.5	23.5	**476317**	476238.4	116.1	**476317**	476305.3	101.0	**476317**	**476317.0**	0.7
	11000	515081	514653	514360.1	22.3	514638	514455.2	125.6	**515081**	515072.1	78.1	**515081**	**515081.0**	50.2
	12000	552714	552640	552195.7	25.1	552576	552382.5	121.7	**552714**	552712.5	40.8	**552714**	**552714.0**	12.2
	13000	590698	590586	590293.4	66.7	590356	590271.6	127.4	**590698**	590696.8	94.5	**590698**	**590698.0**	10.8
	14000	628776	628635	628300.7	38.2	628600	628423.0	113.5	**628776**	628772.4	72.0	**628776**	**628776.0**	5.3
	15000	665158	664999	664557.4	28.3	664957	664479.3	110.0	**665158**	665155.4	98.4	**665158**	**665158.0**	6.9
	16000	703568	703559	703016.0	41.3	703494	703071.4	124.2	**703568**	703564.8	79.4	**703568**	**703568.0**	1.7
	17000	740259	739892	739371.9	20.0	740138	739558.2	130.2	**740259**	**740259.0**	15.8	**740259**	**740259.0**	2.6
	18000	777247	**777247**	777031.2	21.9	777070	776922.8	123.3	**777247**	**777247.0**	20.1	**777247**	**777247.0**	4.1
	19000	813828	**813828**	813609.9	49.2	813628	813345.8	123.2	**813828**	**813828.0**	6.3	**813828**	**813828.0**	1.1
	20000	849829	849817	849681.0	50.5	849817	849682.1	91.1	**849829**	**849829.0**	23.7	**849829**	**849829.0**	3.9

Table A.10: Detailed results of each instance in SET I-dense ($|V| = 1000$) obtained by RL-BLS (this work) and the reference algorithms BVNS, OVNS, OBMA.

k	$\|E\|$	Updated BKS	BVNS			OVNS			OBMA			RL-BLS		
			f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
300	100000	6393066	**6393066**	6387198.3	76.1	6391194	6385309.3	49.7	**6393066**	**6393066.0**	2.1	**6393066**	**6393066.0**	16.4
	110000	6951250	6949241	6942991.0	47.0	6945866	6944676.7	49.2	**6951250**	**6951250.0**	4.1	**6951250**	**6951250.0**	33.1
	120000	7506010	**7506010**	7503678.7	69.2	**7506010**	7503224.6	38.9	**7506010**	**7506010.0**	7.1	**7506010**	**7506010.0**	64.4
	130000	8055169	**8055169**	8052062.9	64.5	**8055169**	8053035.4	63.8	**8055169**	**8055169.0**	2.7	**8055169**	**8055169.0**	3.6
	140000	8623729	**8623729**	8620495.1	67.6	**8623729**	8622943.3	49.3	**8623729**	**8623729.0**	4.2	**8623729**	**8623729.0**	6.1
	150000	9160119	9159815	9158925.6	37.9	**9160119**	9159751.4	28.4	**9160119**	**9160119.0**	2.5	**9160119**	**9160119.0**	0.8
	160000	9652692	**9652692**	9650099.0	97.5	**9652692**	9650391.3	45.8	**9652692**	**9652692.0**	3.1	**9652692**	**9652692.0**	14.6
	170000	10201014	10199777	10193186.4	92.9	**10201014**	10199872.6	101.8	**10201014**	**10201014.0**	1.9	**10201014**	**10201014.0**	5.4
	180000	10670268	10668222	10664047.6	56.5	**10670268**	10667844.0	75.4	**10670268**	**10670268.0**	1.7	**10670268**	**10670268.0**	4.3
	190000	11215084	11210631	11206462.7	56.4	**11215084**	11213888.5	54.0	**11215084**	**11215084.0**	3.0	**11215084**	**11215084.0**	9.7
	200000	11750241	11749076	11743233.9	69.7	11749962	11748864.2	85.7	**11750241**	**11750241.0**	2.3	**11750241**	**11750241.0**	7.2
400	100000	10628685	10626965	10623214.8	61.1	**10628685**	10624792.6	67.8	**10628685**	**10628685.0**	2.6	**10628685**	**10628685.0**	13.8
	110000	11607192	**11607192**	11598451.4	72.0	**11607192**	11605010.8	71.1	**11607192**	**11607192.0**	3.1	**11607192**	**11607192.0**	20.3
	120000	12566643	12565615	12563900.4	51.0	12565615	12563283.4	91.5	**12566643**	**12566643.0**	2.8	**12566643**	**12566643.0**	2.6
	130000	13515871	**13515871**	13513702.4	31.1	**13515871**	**13515871.0**	45.4	**13515871**	**13515871.0**	3.3	**13515871**	**13515871.0**	0.4
	140000	14507805	14507666	14505422.0	36.2	**14507805**	14507479.0	114.8	**14507805**	**14507805.0**	2.9	**14507805**	**14507805.0**	1.7
	150000	15408500	**15408500**	15402193.7	70.1	**15408500**	15408015.7	51.5	**15408500**	**15408500.0**	3.6	**15408500**	**15408500.0**	2.9
	160000	16303527	**16303527**	16298604.1	84.5	**16303527**	16303469.9	88.1	**16303527**	**16303527.0**	2.5	**16303527**	**16303527.0**	6.4
	170000	17264481	**17264481**	17264028.7	85.4	**17264481**	**17264481.0**	73.1	**17264481**	**17264481.0**	1.4	**17264481**	**17264481.0**	1.4
	180000	18082934	18079915	18076738.1	53.7	18081862	18079652.8	92.3	**18082934**	**18082934.0**	2.7	**18082934**	18082826.8	52.4
	190000	19041614	**19041614**	19037486.2	26.0	**19041614**	19039842.9	96.9	**19041614**	**19041614.0**	1.7	**19041614**	19041339.4	30.2
	200000	19997545	**19997545**	19996979.8	76.6	**19997545**	**19997545.0**	46.0	**19997545**	**19997545.0**	1.3	**19997545**	**19997545.0**	0.7
500	100000	15860429	**15860429**	15858863.2	42.2	15859777	15859070.7	79.3	**15860429**	**15860429.0**	4.8	**15860429**	**15860429.0**	3.6
	110000	17338428	**17338428**	17336536.9	47.0	**17338428**	17337665.9	130.7	**17338428**	**17338428.0**	3.8	**17338428**	**17338428.0**	2.7
	120000	18799606	**18799606**	18799537.8	16.2	**18799606**	18799398.4	72.9	**18799606**	**18799606.0**	2.6	**18799606**	**18799606.0**	0.9
	130000	20268024	20267613	20264128.7	28.2	**20268024**	20266865.7	79.7	**20268024**	**20268024.0**	7.7	**20268024**	**20268024.0**	2.0
	140000	21781013	21779016	21777901.3	57.8	**21781013**	21780152.9	93.6	**21781013**	**21781013.0**	2.4	**21781013**	**21781013.0**	3.3
	150000	23158366	**23158366**	23155403.7	11.6	23158360	23157601.4	78.1	**23158366**	**23158366.0**	5.0	**23158366**	**23158366.0**	4.1
	160000	24547333	**24547333**	24546397.4	43.1	**24547333**	24546966.3	69.8	**24547333**	**24547333.0**	1.3	**24547333**	**24547333.0**	4.2
	170000	26028015	26027469	26021352.1	42.2	**26028015**	26028009.2	52.5	**26028015**	**26028015.0**	1.3	**26028015**	**26028015.0**	3.0
	180000	27334048	27332882	27331855.8	54.4	**27334048**	27333226.4	102.5	**27334048**	**27334048.0**	1.7	**27334048**	**27334048.0**	9.6
	190000	28799735	**28799735**	28796392.2	15.9	**28799735**	28797613.6	104.8	**28799735**	**28799735.0**	1.8	**28799735**	**28799735.0**	2.2
	200000	30265043	**30265043**	30263287.7	37.7	**30265043**	**30265043.0**	67.5	**30265043**	**30265043.0**	1.4	**30265043**	**30265043.0**	0.4

Table A.11: Detailed results of each instance in SET II-sparse ($|V| = 3000$) obtained by RL-BLS (this work) and the reference algorithms BVNS, OVNS, OBMA.

k	$\|E\|$	Updated BKS	BVNS			OVNS			OBMA			RL-BLS		
			f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
900	15000	18393644	18303902	18253535.1	353.2	18238899	18214544.5	635.3	18370184	18352678.7	824.6	**18393644**	**18389705.1**	710.4
	18000	20807198	20674098	20618633.6	269.9	20630086	20605027.4	642.7	20756462	20743708.1	854.2	**20807198**	**20799013.5**	651.8
	21000	23424039	23332835	23251358.3	143.1	23319406	23284111.1	649.1	23399708	23387596.2	731.5	**23424039**	**23422580.2**	563.3
	24000	25703678	25644877	25589153.1	154.4	25568685	25548655.4	597.2	25682381	25676874.4	583.7	**25703678**	**25696267.6**	569.2
	27000	27371098	27245055	27126829.6	410.8	27194266	27146615.2	626.0	27355797	27329971.5	685.7	**27371098**	**27363512.8**	649.9
	30000	29634974	29530227	29454795.8	352.4	29481233	29458680.6	682.2	29610484	29597201.2	584.0	**29634974**	**29629687.2**	726.1
	33000	31681912	31571249	31475542.5	200.7	31597926	31510946.8	619.4	31651221	31641926.8	834.1	**31681912**	**31681057.9**	727.6
	36000	33564811	33402421	33334033.8	219.6	33423730	33381709.5	754.6	33542127	33529976.7	731.9	**33564811**	**33564185.6**	744.9
	39000	35530142.8	35351959	35182749.5	395.3	35305257	35248856.7	523.0	35503663	35466809.4	654.1	35517345	**35516608.3**	633.1
	42000	37596921	37476395	37360062.9	394.3	37485555	37430027.9	632.9	37576604	37563791.4	738.3	**37596921**	**37590914.9**	714.3
	45000	39635485	39520397	39423393.5	499.8	39513872	39465558.5	612.8	39628630	39617224.8	680.4	**39635485**	**39629401.7**	526.2
	48000	41429794	41217973	41154418.0	303.0	41235257	41190369.3	617.4	41426227	41413891.1	644.1	**41429794**	**41427369.0**	547.4
	51000	43694897	43554333	43450141.9	418.5	43557180	43496333.6	595.9	43693618	43679024.0	778.8	**43694897**	**43692006.0**	580.6
	54000	45602503	45442741	45363923.8	297.7	45459044	45422669.3	606.5	45601021	45592969.4	551.2	**45602503**	**45602503.0**	130.5
	57000	47514530	47387828	47314394.0	270.2	47464325	47337579.3	566.1	47510249	47495772.4	575.1	**47514530**	**47514530.0**	317.1
	60000	49298989	49175471	49096813.8	263.6	49185527	49103931.6	591.8	49294255	49290202.5	650.0	**49298989**	**49298710.9**	416.4
1200	15000	26303753	26241813	26192984.1	606.1	26111926	26104577.0	183.7	26270826	26232748.9	969.3	**26303753**	**26300694.1**	556.5
	18000	30126273	30056768	30010902.8	216.6	29979897	29939577.7	417.6	30095251	30075020.2	935.0	**30126273**	**30120090.2**	597.4
	21000	34089304	33973726	33944042.7	216.7	33941207	33923510.3	535.4	34064968	34051934.2	773.0	**34089304**	**34087540.5**	689.2
	24000	37741585	37687032	37638223.8	260.1	37635495	37602214.9	674.4	37733227	37720502.0	748.5	**37741585**	**37737317.1**	463.2
	27000	40506553	40408122	40323609.0	359.3	40312262	40273863.2	632.5	40493577	40477297.8	857.5	**40506553**	**40501079.6**	315.6
	30000	44100198	44014313	43947636.9	398.3	43909982	43880438.9	707.9	44083538	44071469.1	727.0	**44100198**	**44091626.1**	431.6
	33000	47370452	47211715	47165807.2	341.5	47210088	47181686.0	554.3	47354678	47344382.8	688.7	**47370452**	**47366845.8**	281.5
	36000	50527695	50473421	50388115.7	303.6	50395619	50362681.9	489.8	50520567	50509235.9	670.9	**50527695**	**50525377.3**	457.9
	39000	53753648	53569869	53482636.1	550.6	53561374	53520065.2	445.2	53729561	53709043.8	770.8	**53753648**	**53747516.9**	474.5
	42000	57035529	56899983	56772672.1	406.3	56858353	56801206.7	654.1	57034960	57017665.1	581.8	**57035529**	**57031797.3**	255.6
	45000	60291909	60220161	60094205.2	400.4	60114348	60055973.4	426.1	60284339	60263415.3	668.0	**60291909**	**60288537.6**	573.1
	48000	63236043	63077933	62966391.6	415.4	63032112	62995113.1	712.7	63229752	63192125.7	733.7	**63236043**	**63230742.0**	647.9
	51000	67034313	66952867	66808693.2	351.1	66904295	66884094.7	649.3	67027129	67018688.1	652.2	**67034313**	**67033342.2**	590.2
	54000	70148933	70042644	69985967.4	359.8	70033332	69990863.2	608.9	**70148933**	70138870.5	675.2	**70148933**	**70148520.3**	432.1
	57000	73089013	73019678	72959255.8	412.1	72965508	72937927.5	589.6	**73089013**	73083073.6	708.3	73087757	**73086302.2**	429.2
	60000	76038735	75953006	75871889.6	424.0	75889026	75804127.9	434.4	76037257	76033548.2	442.8	**76038735**	**76038357.0**	449.8
1500	15000	34734273	34675986	34636108.3	646.1	34574592	34554733.3	505.6	34713826	34674083.5	972.4	**34734273**	**34727378.3**	299.9
	18000	40210545	40161374	40123088.9	229.0	40033168	40008802.9	454.7	40176240	40164162.9	944.0	**40210545**	**40208820.2**	327.2
	21000	45705061	45629510	45599244.5	201.5	45575395	45550898.1	560.1	45680921	45664014.2	814.8	**45705061**	**45697297.1**	397.9
	24000	50916622	50874877	50816341.1	175.9	50787821	50748683.6	461.9	50902394	50890876.8	708.6	**50916622**	**50915824.5**	380.6
	27000	55141900	55028345	54991102.7	493.6	54940568	54915119.2	449.9	55128559	55111526.0	774.2	**55141900**	**55137287.0**	88.4
	30000	60176308	60093139	60055683.7	235.1	60020987	59974641.5	488.6	60172745	60148641.9	675.6	**60176308**	**60169428.0**	313.0
	33000	64937647	64869295	64805487.5	206.1	64782561	64752239.9	553.5	64927552	64910716.9	742.4	**64937647**	**64929037.3**	505.3
	36000	69653450	69586603	69524997.7	554.4	69522884	69497007.5	543.9	69639543	69632330.8	822.8	**69653450**	**69648684.2**	155.5
	39000	74509046	74400594	74347341.2	595.5	74375244	74353914.2	500.4	74507275	74494263.2	658.0	**74509046**	**74501678.5**	115.6
	42000	79189422	79059927	78973488.5	485.7	78967740	78944098.5	546.6	79182896	79173305.9	685.8	**79189422**	**79187731.1**	354.7
	45000	83831097	83779672	83635929.7	505.9	83674776	83611129.7	533.2	83818836	83805548.4	762.6	**83831097**	**83824290.9**	392.9
	48000	88216978	88098004	88020315.5	468.0	88018953	87986803.2	386.3	88207316	88194058.9	776.3	**88216978**	**88203578.6**	340.0
	51000	93804870	93754203	93678542.2	469.9	93709555	93649840.9	470.3	93802469	93797106.0	600.8	**93804870**	**93800865.7**	616.2
	54000	98363598	98302971	98219848.4	271.1	98219667	98200329.9	493.7	**98363598**	**98360782.5**	570.2	98362235	98360152.3	293.4
	57000	102567225	102519095	102429911.4	301.7	102393003	102358842.1	605.1	102563339	102559232.1	486.5	**102567225**	**102564898.6**	661.4
	60000	106939101	106906241	106810293.8	308.8	106812476	106782137.2	402.2	106938166	106935416.4	591.1	**106939101**	**106937824.3**	472.1

Table A.12: Detailed results of each instance in SET III-sparse ($|V| = 1000$) obtained by RL-BLS (this work) and the reference algorithms BVNS, OVNS, OBMA.

k	$\|E\|$	Updated BKS	BVNS			OVNS			OBMA			RL-BLS		
			f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
300	5000	2505650	2493324	2488171.4	125.3	2469693	2460143.1	114.4	2465929	2454227.2	136.8	**2505650**	**2502485.4**	17.3
	6000	2468480	2446500	2440413.8	142.5	2440956	2430658.2	114.1	2431299	2427970.2	133.2	**2468480**	**2464962.1**	19.4
	7000	2565139	2552404	2548214.9	110.8	2545201	2542584.4	111.5	2534746	2529880.1	142.1	**2565139**	**2560552.5**	43.4
	8000	2651401	2579864	2554365.8	111.6	2615323	2611577.7	59.7	2635788	2630477.4	149.1	**2651401**	**2648962.0**	86.7
	9000	2706738	2590223	2552834.8	52.0	2673069	2669126.0	69.9	2676197	2669249.3	125.5	**2706738**	**2703789.6**	51.9
	10000	2757724	2591920	2542248.7	104.3	2736823	2729978.9	110.9	2736197	2731101.7	151.4	**2757724**	**2755845.0**	12.5
	11000	2795850	2707028	2680823.8	75.0	2766499	2757954.4	63.9	2768254	2762756.9	128.7	**2795850**	**2794057.5**	21.4
	12000	2799009	2704948	2671120.3	63.1	2769381	2762476.5	99.3	2777104	2771728.9	120.5	**2799009**	**2796537.5**	20.9
	13000	2861878	2656264	2628160.8	105.6	2835967	2830483.2	131.8	2839769	2832137.7	176.9	**2861878**	**2861477.3**	43.0
	14000	3007395	2875145	2848269.4	67.5	2974290	2970998.6	84.9	3000167	2990203.3	145.5	**3007395**	**3006410.3**	12.6
	15000	3025684	2954952	2934727.3	103.3	2998564	2994337.3	86.4	3008260	2998903.7	117.0	**3025684**	**3023935.4**	27.1
	16000	3159367	3035161	3025875.1	112.8	3132375	3125121.8	73.4	3151327	3144060.2	145.8	**3159367**	**3159167.6**	3.4
	17000	3095759	2943051	2859287.1	121.1	3077285	3070048.7	92.9	3070689	3062003.9	153.1	**3095759**	**3094463.7**	15.8
	18000	3142925	3040885	3001286.7	99.2	3131940	3124574.0	85.5	3125677	3118708.6	116.2	**3142925**	**3142381.3**	23.3
	19000	3234195	3170547	3156263.3	109.9	3218385	3216043.3	111.0	3224836	3221268.2	148.9	**3234195**	**3234195.0**	3.2
	20000	3301025	3134058	3053655.5	90.2	3278104	3269857.5	130.8	3287716	3280517.1	124.3	**3301025**	**3300250.9**	32.8
400	5000	3586663	3574084	3561188.5	72.6	3533213	3528853.9	75.6	3548101	3532630.3	162.4	**3586663**	**3584789.4**	88.4
	6000	3695671	3679838	3674254.7	116.7	3667943	3662150.4	85.4	3676972	3663648.6	147.6	**3695671**	**3694636.8**	81.9
	7000	3814597	3793994	3786512.6	88.6	3783655	3781743.5	121.6	3803268	3796320.8	144.0	**3814597**	**3813389.7**	97.4
	8000	3883149	3675546	3653571.1	80.5	3834856	3824087.2	124.9	3869853	3866171.7	134.1	**3883149**	**3882707.4**	80.1
	9000	3992275	3942894	3896982.2	104.3	3957694	3944602.4	139.9	3978835	3973985.1	144.2	**3992275**	**3990693.2**	53.4
	10000	4131119	4078596	4056753.0	96.0	4095384	4089169.9	111.7	4121055	4117785.9	146.0	**4131119**	**4130787.8**	37.5
	11000	4182178	4143927	4086375.0	56.8	4145325	4139732.9	86.2	4166026	4162338.9	161.4	**4182178**	**4181385.1**	101.8
	12000	4292154	4194556	4133244.5	130.3	4244091	4237711.9	112.9	4286109	4277275.1	143.9	**4292154**	**4292109.1**	54.6
	13000	4441534	4285580	4256425.3	19.4	4410499	4405131.0	103.0	4425963	4422160.2	135.8	**4441534**	**4440038.8**	32.6
	14000	4440159	4325063	4284824.2	116.0	4407778	4401470.3	147.3	4435509	4426989.2	165.8	**4440159**	**4439943.2**	51.8
	15000	4577660	4480464	4437816.0	66.5	4540691	4535941.1	86.3	4556198	4553281.3	149.5	**4577660**	**4577016.2**	64.1
	16000	4604394	4498498	4464807.9	86.9	4571827	4561015.0	142.1	4595152	4591261.5	140.2	**4604394**	**4604115.2**	49.8
	17000	4672641	4612420	4528612.6	62.2	4644941	4640741.9	109.7	4658010	4651214.0	143.9	**4672641**	**4672364.6**	62.0
	18000	4874088	4739060	4684688.3	110.4	4837488	4832091.3	107.0	4860486	4856466.3	160.0	**4874088**	**4872625.7**	58.8
	19000	4898657	4808972	4783438.0	56.0	4857941	4854705.1	90.5	4884685	4876382.4	144.5	**4898657**	**4898571.9**	56.9
	20000	5069605	5007301	4987676.8	109.0	5048499	5039748.5	134.1	5057136	5049774.0	153.3	**5069605**	**5069287.4**	31.7
500	5000	4848262	4830650	4818463.4	85.6	4813388	4809096.9	55.1	4823713	4813298.0	161.1	**4848262**	**4846674.1**	90.9
	6000	5034263	5009081	4998333.2	120.0	4980835	4976986.4	113.6	5006202	5001071.1	170.6	**5034263**	**5034179.2**	98.0
	7000	5165747	5149458	5138705.1	145.0	5119885	5110429.7	61.4	5155501	5151256.5	148.3	**5165747**	**5165432.6**	57.2
	8000	5338298	5267348	5162209.2	27.2	5275144	5262995.8	49.6	5330241	5327346.3	126.7	**5338298**	**5338296.0**	21.2
	9000	5414321	5358740	5308557.8	48.9	5367860	5363246.8	125.7	5405874	5403362.2	163.1	**5414321**	**5414202.2**	60.9
	10000	5569556	5437888	5391829.8	50.9	5528846	5520846.8	90.1	5559264	5555544.6	144.6	**5569556**	**5569464.6**	79.4
	11000	5713899	5580829	5562160.8	13.5	5650209	5644264.8	89.5	5699047	5696247.8	171.8	**5713899**	**5713756.9**	92.0
	12000	5869809	5696150	5670367.2	61.7	5828407	5821463.2	103.1	5852876	5848883.7	119.1	**5869809**	**5869211.9**	79.3
	13000	6017145	5883983	5806531.4	51.6	5984228	5973749.9	106.2	6004154	5994977.4	153.6	**6017145**	**6016862.4**	33.1
	14000	6215608	6106153	6082426.3	79.3	6180484	6172030.1	79.2	6207369	6204912.5	157.4	**6215608**	**6215608.0**	65.7
	15000	6290565	6127240	6098089.0	36.5	6251991	6249809.7	74.6	6283821	6281639.9	118.1	**6290565**	**6290360.0**	128.5
	16000	6439191	6373991	6356911.1	56.6	6393207	6390426.4	105.8	6427512	6423626.0	149.8	**6439191**	**6439101.5**	77.1
	17000	6643825	6417358	6390650.7	33.8	6606993	6598223.8	89.1	6632166	6629103.9	156.7	**6643825**	**6643825.0**	44.7
	18000	6769342	6640806	6597486.3	53.3	6748356	6744221.5	104.0	6763646	6762603.4	149.0	**6769342**	**6769148.4**	59.7
	19000	6885340	6789379	6755338.3	61.8	6851741	6845734.7	69.0	6876847	6872393.6	143.2	**6885340**	**6885340.0**	51.4
	20000	7092854	7051655	7013729.8	87.3	7065237	7060466.2	98.6	7084264	7081940.0	158.5	**7092854**	**7092854.0**	42.5