JOURNAL OF INDUSTRIAL AND
MANAGEMENT OPTIMIZATION
Volume **17**, Number **6**, November **2021**

doi:10.3934/jimo.2020115

pp. **3223–3246**

TABU SEARCH GUIDED BY REINFORCEMENT LEARNING FOR THE MAX-MEAN DISPERSION PROBLEM

Dieudonné Nijimbere
Songzheng Zhao, Xunhao Gu and Moses Olabhele Esangbedo*

School of Management
Northwestern Polytechnical University
710072, Xi'an, China

Nyiribakwe Dominique

Department of Computer science and Technology
Xidian University
710071, Xi'an, China

(Communicated by Oleg Prokopyev)

Abstract. We present an effective hybrid metaheuristic of integrating reinforcement learning with a tabu-search (RLTS) algorithm for solving the max–mean dispersion problem. The innovative element is to design using a knowledge strategy from the Q-learning mechanism to locate promising regions when the tabu search is stuck in a local optimum. Computational experiments on extensive benchmarks show that the RLTS performs much better than state-of-the-art algorithms in the literature. From a total of 100 benchmark instances, in 60 of them, which ranged from 500 to 1,000, our proposed algorithm matched the currently best lower bounds for all instances. For the remaining 40 instances, the algorithm matched or outperformed. Furthermore, additional support was applied to present the effectiveness of the combined RL technique. The analysis sheds light on the effectiveness of the proposed RLTS algorithm.

1. **Introduction.** Suppose a set V with n vectors and $[d_{ij}]_{n\times n}$, a distance matrix where d_{ij} determines the length from element i to j, and the distance can be taken as both a positive and a negative value. The max–mean dispersion problem consists of picking a subset M of m vectors from V ($|M|$ is not fixed) so that the average distance between selected vectors, i.e., $\left(\sum_{i,j\in M;i<j} d_{ij}\right)/|M|$, is maximized. Assume that variable x_i equal to 1 when the vector is picked in M, and equal to 0 otherwise; then, the max–mean dispersion problem can be formally considered as a local binary quadratic programming problem first as follows:

$$\text{Maximize } f(s) = \frac{\sum_{i=1}^{i=n-1}\sum_{j=i+1}^{j=n} d_{ij}x_ix_j}{\sum_{i=1}^{n} x_i} \tag{1}$$

2020 *Mathematics Subject Classification.* Primary: 65K05; Secondary: 90-08.

Key words and phrases. Hybrid metaheuristics algorithm, reinforcement learning, Q-learning algorithm, tabu search.

* Corresponding author: Moses Olabhele Esangbedo.

Subject to

$$\sum_{i=1}^{n} x_i > 2 \tag{2}$$

$$x_i \in \{0,1\}, i = 1,2,...,n \tag{3}$$

Constraints in Equations (2) and (3) state that at least two variables are selected.

Dispersion problems are distinguished in the literature in term of the equity and efficiency distance function based on [22]. The objective function of the max–mean dispersion problem is accordingly taken as equity-based. Several other specific dispersion problems can be found and defined in the literature include: max-mean dispersion problem that maximize the mean of the aggregate dispersion, the max-min diversity problem that maximize the minimum dispersion [10, 21], the minimum differential problem that minimize the difference between the maximum aggregate dispersion and minimum aggregate dispersion [1, 26],the balanced quadratic optimization problem that minimizes the difference between the maximum dispersion and the minimum dispersion [23], a Solution-based tabu search for the maximum min-sum dispersion problem characterized by the joint use of hash functions to determine the tabu status of candidate solutions and a parametric constrained swap neighborhood to enhance computational efficiency [14] etc. Another efficiency-based dispersion problem is max-sum dispersion that maximizes the aggregate dispersion [2, 27]. Furthermore, one of the well-known equity-based diversity problems is the max–mean dispersion problem (max–mean DP) and weighted version of max–mean DP [15, 20], when selected subset M consists of a flexible cardinality, i.e., the range of subset M can be from 2 to n. This paper focuses on this last specified max–mean DP, and aimed at designing and implementing effective Metaheuristic algorithms for dealing with it. In addition to its theoretical significance as an NP-hard problem [22], the max–mean dispersion problem has practical applications in various fields, such as architectural-space planning, social-relations mining, web-page ranking, pollution control, and capital investment.

Several algorithms have been presented in the literature to solve difficult optimization problems. In 2009, Prokopyev et al. [22] proposed the linear mixed binary programming formulation for the max–mean dispersion problem, and proved that the max–mean dispersion problem is NP-hard where the distance between elements is not limited. Xu et al. [28] applied learning adaptation to solve constraint-satisfaction problems. A hybrid metaheuristic approach based on reinforcement learning that is applied to the traveling-salesman problem was proposed by Francisco et al. [7] in 2010. In that same year, Tao and Zhen [29] also introduced a multistep $Q()$ learning approach to power-system stabilizers. In 2013, Martin and Sandoya proposed the Greedy Random Adaptive Search integrated with the Path Relinking Method (GRASP-PR) [18]. It uses a randomized greedy mechanism to maintain first building elite solutions (ES) and a variable neighborhood descent procedure for improvement. In 2014, Croce et al. [8] developed a two-stage hybrid heuristic method that combines a mixed integer nonlinear solver and a local branch program. Late in 2016, the same author improved a new version of this algorithm by adding a path phase enhance the quality of the solution [9]. In 2015, Carrasho et al. [6] dynamically used an efficient two-savage tabu-search algorithm. The strategy combined various short-term and long-term tabu searches to improve their performance. Brimberg et al. [4] introduced a search algorithm method that examines multiple neighborhood structures based on add, drop, and swap moves

and picks one of them in a probability way to perform a shaking procedure. In 2016, Lai and Hao [15] developed the first tabu-search population-based memetic algorithm (MAMMDP). The algorithm first uses a tabu search to detect promising local optima and then randomly a crossover operator procedure to create better diversified offspring solutions.

After reviewing the above algorithms, it is worthwhile to note that the algorithms after 2013 are more contrastive. Another important component in our algorithm is the reinforcement-learning mechanism is an unsupervised learning algorithm. Since it does not need an experienced agent in choosing the exact way; instead, it performs its future actions based on an obtained feedback response from the environment [13].

The three most distinct elements of an RL agent are state, actions, and rewards. RL functions in the same way as solving a heuristic optimization problem. Miagkikh and Tunch [19] proposed a population of RL agents to deal with combinatorial optimization problems. The state of cellular automata applied to the graph-coloring problem was updated using RL [25]. An interacting application of RL is found in its combination with metaheuristics [3]. For example, RL is applied to learn a new evaluation function over multiple search trajectories of the same problem instance and alternates between using the learned and the original evaluation function. For this, Xu et al. [28] proposed a formulation of constraint-satisfaction problems as an RL task. Indeed, it can be used in choosing heuristics among the many available, and knows when to use it. Another use of RL arises in hyperheuristics, where it a combines section strategy with an acceptable strategy for choosing suitable low-level heuristics and deciding when to agree on a move. For example, Burke et al. [5] proposed hyperheuristics in which the selection of low-level heuristics makes use of basic reinforcement-learning principles combined with a tabu-search mechanism. More recently, RL is used to schedule several search operators under genetic and multiagent-based optimization frameworks. Another example can be found in the works of Sghir et al. [24]. The two mechanisms above rely on the use of RL approaches instead of the random component to obtain a more knowledgeable decision strategy. In this paper, reinforcement-learning-based local search (RLTS) uses an RL mechanism to build promising initial solutions for iterative Q-learning to directing a tabu search. Thus, we present the RLTS approach for tackling the max–mean DP, which combines reinforcement-learning techniques with a dynamic tabu-search procedure (Section 3).

Our proposed (RLTS) algorithm belongs to a multistart search framework where two main components, at each start, include an initial solution-building mechanism and a local-search mechanism. For the initial solution-generation phase, we were inspired by the idea of the Q-Learning strategy that uses an informed-decision strategy from a set of agents interacting in the environment to build high-quality and well-diversified solutions. Then, we used a one-flip tabu search algorithm for the local phase to process the initial solution, to improve the quality of solutions, and obtain the feasible approximated optimal solution. Computational experiments on extensive benchmarks showed that the RLTS performed much better than state-of-the-art algorithms in the literature. Furthermore, additional support was applied to present the merit of the combined RL technique. Next, the paper is organized as follows. Section 2 describes the general procedure and important components of the proposed RLTS algorithm. Section 3 presents our computational results, comparisons with state-of-art algorithms, and analyzes the contributions of the RL

component of our algorithm to its performance. Conclusions based on computational outcomes are given in Section 4.3.

2. **Algorithm.** In this section, the proposed general RLTS algorithm scheme, feasible-set initialization mechanism, RL guided solution-generation technique, tabu-search strategy, feasible-set updating, and reconstruction methods are presented.

2.1. **General scheme.** The main scheme of our RLTS algorithm is depicted as Algorithm 1. The algorithm starts by constructing the relevant feasible set and initializes it (Section 2.3), then enters the loop (Lines 5–18). For the first cycle, an initial solution S_0 is randomly constructed (Section 2.4). Then, initial solution S_0 is processed into a current solution S_c using a tabu-search algorithm (Section 2.3). According to whether the element in initial solution S_0 is still present in current solution S_c or whether current solution S_c is greater than current optimal S^*, we update the reward matrix so as to increase the reward of the element still present or reduce the reward of the element that no longer exists (Section 2.4). The next step is to decide whether current solution S_C is greater than current optimal solution S^*. If S_c is greater, it replaces current optimal solution S^* (Lines 14–15), and then enters the next cycle. Starting from the second cycle, some feasible set needs to be initialized, but note that the Q matrix, the reward matrix, and current optimal solution S^*, do not participate in initialization (Section 2.2.1). Apart from using a reinforcement-learning algorithm to generate the initial solution to the tabu-search algorithm for later processing (Section 2.2.2), the other processes remain unchanged. At the end of each loop, it again decides whether or not the current running time exceeded the limited time. Otherwise, it does not enter the next cycle and produces current optimal solution S^* (Lines 3, 17–19).

Algorithm 1 : General scheme of RLTS Algorithm.

1: **Input:** Number of elements in set V: Varnum, instance matrix $D = [d_{ij}]_{n\times m}$, timeout limit : t_{limit};
2: **Output:** Best solution S^*;
3: $t_1 \leftarrow$ Time () ;
4: InitData_start(); /*Section 2.2.1 */
5: **loop**;
6: InitData(); /*Section 2.2.1 */
7: **If** first loop, then
8: $S_0 \leftarrow$ Random () ; /*Section 2.2.1 */
9: **Else**;
10: $S_0 \leftarrow$ Reinforcement learning () ; /*Section 2.2.2 */
11: **end if**;
12: $S_c \leftarrow$ Tabu search (S_0) ; /*Section 2.3 */
13: RewardUpdate(S_0,S_c,S^*); /*Section 2.4 */
14: **If** $f(S_c) > f(S^*)$ then
15: $S* \leftarrow S_c$;
16: **End if**;
17: $t_2 \leftarrow$ Time () ;
18: **End loop** if $(t_2 - t_1) > t_{limit}$;
19: **Output** S^*.

2.2. **Feasible-Set initialization.** To construct the RL Model, we started by initializing a feasible set (FS) to generate a set of good-quality solutions. To build each solution in FS, we first used distance matrix D to build a reward matrix; then, we constructed the Q-learning matrix (QLM) on the basis of reward-matrix information. Meanwhile, the FS set to be initialized every loop in Feasible Initialization Set () records the three components: initial solution S_0, current solution S_0, and the tabu table.

2.2.1. *Randomly Constructed Initial Solution.* At the beginning, the algorithm records reward matrix R to be a zero matrix if the reinforcement algorithm is used to construct the initial solution. At this time, each action selection is not rewarded, which leads to a feasible set of solutions in the Q matrix, which implies that the resulting experience matrix cannot be updated. Hence, Q-matrix is likewise set to be a zero matrix. If reinforcement learning constructs the first initial solution, it may try all currently available actions, that is, the initial solution selects all elements. Such an initial solution is to be fixed, and records initial solution S_0, current solution S_C, and updated current optimal solution s^*. Updating reward matrix R is supposed to be fixed in the first cycle. Therefore, in order to diversifying the search move of the algorithm, the initial solution of the first cycle is Random () performed. The algorithm assigns each variable with the equal probability of 0.5 to receive value 1 or 0. Then, a one-flip move operator is applied by a tabu search for improvement.

2.2.2. *RL Initial-Solution Construction.* After going through the first cycle, by comparing initial solution S_0 with current solution S_c, it is already possible to define a reward of selecting some elements. The main idea of rewards in this algorithm is that if the element in initial solution S_0 still arrives in current solution S_c after tabu-search refinement, it is considered to be a superior element. If this element is selected, it is more promising to increase the value of the objective function and shorten tabu-search-use time. However, a positive return is defined for all actions that select this kind of element. On the other hand, if the element in initial solution S_0 does not exist in current solution S_c after tabu improvement, then all the actions of selecting this element record a negative return of the same value. Because the reinforcement-learning algorithm has a higher probability of selecting the action with the highest return when selecting an action, the former has a greater probability of being selected, and the probability of the latter to be selected reduces.

At the same time, if the value of the objective function of current solution S_c is greater than current optimal solution S^*, it means that elements in current solution S_c are even better; then, the value of the reward is accordingly increased (Section 2.2.3). By doing so, the elements in current optimal solutions S^* are more likely to be selected in the initial solution. Although this can save a lot of work for tabu searches and shorten time, the initial solution has very strong similarities, and it is easy for the optimal solution found by the algorithm to become trapped in a local optimal contradiction. Therefore, the action-selection strategy does not always choose the action with the highest reward; instead, some probabilities choose random actions to improve search diversity (Section 2.2.4). In this way, we can ensure that the initial solution generated by the reinforcement-learning algorithm is better. The specific steps for reinforcement learning to build an initial solution, detailed in Algorithm 3, are as follows:

1. Initialize optional-action list;
2. randomly select initial state;
3. update optional-action list;
4. select action according to selection strategy;
5. recognize reward from R matrix according to state and action;
6. compute state–action function $Q(s, a)$ according to formula ;
7. determine whether or not to meet the termination condition according to the obtained state–action function value. If it meets, outcome initial solution; if not, update Q matrix, perform the action, enter the next state, and then loop Steps 3 to 7;

Algorithm 2. Reinforcement Learning ()

1: **Input**: number of elements in met V; optional action list S_a; reward matrix R; experience matrix Q;
2: **Output**: initial solution S;
3: **For** $i = 1$ to n do;
4: Initialize optional action list l () ;
5: Randomly select initial state;
6: $S[state - 1] = 1$;
7: **Loop**;
8: Update optional-state action; /* Section 2.2.3 */
8: Select action according to selection strategy; /* Section 2.2.4 */
9: Recognize reward from R matrix according to state and action;
10: compute state-action function $Q(s, a)$ according to Equation (4);
11: Verify termination conditions (Equations (5) and (6)); /* Section 2.2.5*/
12: **Break**;
13: $S[action - 1] = 1$;
15: $Q[state - 1, action - 1] = Q'$;
16: $state = action$;
17: **End loop**;
18: **Output** S.

2.2.3. *Optional-Action List Update.* The reinforcement-learning algorithm first aims to construct an empty set, and then continuously selects the action to put the element into the set without removing elements from the set. To avoid selecting actions that have already been selected, an array of optional-state actions S_a are built into the algorithm. The array stores currently selectable actions. At the beginning, the size of the array is the size of the problem, that is, number of all elements n, for example, $S\_a[0] = 1$, $S\_a[1] = 2$, ..., $S\_a[n - 1] = n$. The array records counter S_aNum representing the number of current optional actions; initial size is n. After random selection of the initial state and each action-selection, array S_a is updated. Specifically, it removes the element selected by the operation from $S\_a[\ ]$, then all numbers after this element are forwarded by one square, and counter S_aNum is decremented by one. In this way, it is possible to avoid wasting time by selecting actions that have already been selected and, to a certain extent, improve the search efficiency of the algorithm.

2.2.4. *Action-Selection Procedure.* At each step, the reinforcement-learning algorithm selects actions from the optional-action array according to the action-selection strategy. There are generally four action-selection strategies that were detailed by Zhou et al. [30]: random selection, greedy selection; roulette selection based on the values of experience matrix Q; and hybrid selection, which mixes random-selection and greedy-selection strategies. The choice of action-selection strategy is very important for the reinforcement-learning algorithm. Therefore, our proposed algorithm uses a hybrid selection strategy to select actions that combine randomness with greediness search. When the greedy strategy is adopted with the probability of ϵ, a seemingly optimal element is selected, thereby improving the quality of the initial solution, speeding up the search for the optimal solution by tabu search, and at the same time exploiting the action to make the model more accurate. When using a random strategy with a probability of 1 - ϵ, it randomly selects an optimal element, so that the algorithm occasionally avoids greed, reduces the degree of similarity of the initial solutions, and reduces the probability of falling into trapped local optima. At the same time, it can explore unperformed actions and dynamically develop more possibilities to make the strategy more comprehensive.

2.2.5. *Termination-Criterion Conditions.* In reinforcement-learning procedures, the goal is to average over the results from state actions, so we keep a running average, and then updating the value under a free model, or without a model, when an agent travels a trajectory distance linking state S to state S', which can be found using action a, performed using the iteration formula for state-action function Q(s,a), as follows [6]:

$$Q\left(St, at\right) = \left(1 - \alpha\right) Q\left(St, at\right) + \alpha\left(r + \gamma maxQ\left(S_{t+1}, a_{t+1}\right)\right), \tag{4}$$

where S_t is the current action, a_t is the current state, S_{t+1} is the next state,a_{t+1} is the next action, α is the learning factor, $0<\alpha<1$, r is the reward to choose this current action, γ is the discount factor, $0\leq$ r<1, and max $Q(S_{t+1},a_{t+1})$ is the maximum reward available for the next state. Therefore, after many iterations, r can be regarded as the current reward, γ max Q(S_{t+1},a_{t+1}) is the maximum future reward after discounting, i.e., $r+\gamma$ max Q (S_{t+1},a_{t+1}) is the largest reward for the current action; $(1-\alpha)Q(S\_t, a_t) + \alpha(r + \gamma$ max $Q(S\_t + 1, a_{t+1}))$ is the comprehensive maximum reward combining past experience and current realities. Then, the termination condition in this algorithm is given by the following formula:

$$Q' < (1-\alpha)Q[state - 1, action - 1] \tag{5}$$

If this condition is met, selecting this action leads to a reduction in the comprehensive maximum reward. However, the purpose of reinforcement learning is to maximize the comprehensive maximum reward as much as possible. If these two are inconsistent, the algorithm is terminated. Another termination condition of this algorithm can be found when reaching the condition as:

$$Q[state - 1, action - 1] \neq 0. \tag{6}$$

this condition indicates that the training Q matrix has already had the experience of selecting this action in the state, and this can effectively reduce loss of the fact that an excellent element is no longer present in some local optimal solution, it is therefore selected by the very low probability in the later period. Finally, the reinforcement-learning algorithm terminates only when the following two conditions

are met. Let $Q = Q$[state-1, action-1] then,

$$Q' < Q \&\& Q \neq 0. \tag{7}$$

2.3. **Tabu-search procedure.** In the case of the max–mean dispersion problem, tabu searches have been largely used with great success in the literature, as was proven by Lai and Hao [15], as well as Carrasco et al. [6]. In particular, the MAMMDP algorithm proposed by Lai and Hao [15] is still the current optimal algorithm. Therefore, the tabu search designed in this paper is the same as the one first used by them [15]. The main difference is the effect of constructing the initial solution using reinforcement learning instead of the genetic algorithm. The process of a tabu search improving the initial solutions in this algorithm is mainly performed by interacting through its main components, including move operator, objective-function assessment, selection of solution mechanism, and tabu aspiration criteria. To interchange the current solution into its neighbor's solution, a one- flip move operator is used. This strategy of a one-flip move has been largely used in dealing with various binary quadratic programming problems [17]. The assessment of this function is referred to the objective function to measure solution quality. For each iteration, the best improvement mechanism is applied among neighborhood solutions where the best evaluation-function value is more biased.

According to the size of the problem (n variables), the size of the neighborhood solutions is $O(n)$; then, the computational process time to identify the best solution becomes time-consuming when using Equation (1).when seeking computation simplicity. A quick strategy that evaluates the neighborhood solution is described in [15] to improve the search efficiency of a tabu search.

Practically, we set a one-dimensional array $W = \{c_1, p_2, \ldots, p_t\}$ to compute the number of moves of each possible move applicable to current solution S, where p_i represents the total distance between element i and the other selected elements in the current solution, defined as:

$$P_i = \sum\nolimits_{j \epsilon M, j \# i} d_{ij} \tag{8}$$

where M represents the set of the selected elements. In this way, when the one-flip move flips element x_i to 1-x_i, the amount of the move gain Δ_i can be immediately computed as:

$$\Delta_i = \begin{cases} \frac{-f(s)}{|M+1|} + \frac{p_i}{|M+1|}, & if \quad x_i = 0 \\ \frac{f(s)}{|M-1|} - \frac{p_i}{|M-1|}, & if \quad x_i = 1 \end{cases} \tag{9}$$

Where $f(s)$ is the objective-function value of solution s, and M is the number of selected elements in solution s, that is, the number of elements with a value of 1. After element x_u is flipped, updating one-dimensional array W can be represented as:

$$p_j = \begin{cases} p_j + d_{ij}, if \ x_i = 0, j \neq i \\ p_j, if \ j = i \\ p_j - d_{ij}, if \ x_i = 1, j \neq i \end{cases} \quad \ldots \tag{10}$$

According to the above formula, one-dimensional array W is set to be initialized at the beginning of tabu search. Then, computational complexity becomes $O(n)$, which greatly simplifies the computational complexity of a tabu search and greatly improves search efficiency.

Tabu uses a parameter called tabu tenure to restrict the reverse move from being performed each time a move is performed in the next determined number of tabu tenure. The number of iterations (tabu tenure) are then subdivided into

different intervals recognized according to a specified order of tabu tenure. The above strategy was first introduced by Galinier [11] as a management technique that dealt with the graph-partitioning problem instead of adopting the common static tabu list-management strategy. In this strategy, the true tabu tenure is generated by a periodic transition function consisting of different intervals from the divided iterations. In addition, this strategy was more effective than that of Lai and Hao [15] in solving the max–mean dispersion problem. Therefore, the experiment showed that this strategy can effectively make the tabu-search algorithm achieve a good balance between diversification and intensification. Furthermore, the tabu-search strategy lasts until the best solution reached cannot be improved for a certain number of iterations d, so-called search depth. The specific operation is to set a search-depth counter d at the beginning of tabu search; if the objective-function value of the optimal neighborhood solution found during the iteration is better than the current optimal solution, it means that the iteration is meaningful, and counter d is then cleared. Otherwise, counter d is incremented by one and terminated when counter d has accumulated a specified number of iterations alpha. In doing so, the objective-function value achieved from a one-flip move is found by either the restricted move with the best objective value or a tabu move with an aspiration rule chosen to be performed and selected to join toward the solution.

2.4. **Reward-matrix updating.** After constructing the initial solution and processing it with tabu search, the reward matrix needs to be updated by comparing the initial solution with the current solution to construct a better initial solution in the next iteration. The specific idea behind this is that, if the element in the initial solution can remain in the current solution after being flipped through tabu search, the element is considered to be more interested. Selecting this element is beneficial to approach the global optimum; consequently, this grants it a reward when it is selected. If the element in the initial solution does not remain in the current solution after it has been flipped through a tabu search, then the element is considered to be poor, and it is punished when it is selected. The reward and punishment value is initially set to 1; if the current solution is better than the current optimal solution, then the value of reward and punishment is increased referring to the highest historical reward and punishment value. Therefore, elements that still remain in the current solution are more special; otherwise, the reward and punishment value is reset to 1.

3. Computational results and comparisons.

3.1. **Experiment protocols and benchmark instances.** In this section, we show the experiment results of the RLTS algorithm for 100 benchmark instances and compare them with other algorithms to evaluate RLTS performance. After flipping a variable x_i to be its complement $1 - x_i$, the potential p_j is then updated as: (3.9)

Definitely, by Using the above-mentioned method, the computational complexity of each iteration is reduced to $O(n)$ which is with greater importance when dealing with large scale instances. Tabu rule requires that each time a move is performed, its reverse move is prohibited from being performed during the following specified number of iterations (called tabu tenure). A dynamic tabu tenure management strategy that divides the iterations into different intervals and uses a specified order of different tabu tenures for these intervals is employed. This strategy was first

proposed by Della Croce et al. [10] for solving the graph partitioning problem and demonstrated to be effective for the Max-Mean DP [12]. Furthermore, an aspiration rule overrides the tabu rule if performing a tabu move leads to a better solution than the best solution found so far. The proposed tabu search phase repeatedly carries out the following iterations until the best solution can not be improved for a consecutive number of iterations lamda, called tabu search depth. To be specific, all the moves are first categorized as tabu and non-tabu according to the tabu along with aspiration rules. Then the objective function value of each neighborhood solution caused by the 1-*ip* move is quickly evaluated according to Eq. 11 and 12. Finally, either the non-tabu move with the best objective value or a tabu move that satisfies the aspiration rule is chosen to be performed to move to the next solution.

3.1.1. *Elite set updating and rebuilding.* We employ a popular quality based elite set updating strategy, where the elite set updating happens whenever the newly generated solution gets a better objective value than the worst solution in ES. In this case, the worst solution is replaced by this solution to execute the ES updating procedure. Otherwise, this new solution is disregarded and ES updating fails. When ES does not undergo successful updates for continuous times, we rebuild ES to restart the search. The rebuilding strategy we use is to keep the best solution found so far in ES and generate the other solutions in the same way as the initial ES does.

4. **Computational results and comparisons.** Our aim is to show and demonstrate by Computational results and comparisons between our proposed method with the existing ones. Accordingly,this section is divided into five sub-sections. In the first sub-section, the Benchmarks and experimental protocols are presented. The second sub-section covers the Parameter sensitivity analysis. The third subsuction describes the Experimental results found so far. The fourth sub-section presents the balance between intensification and diversification. Finally, the fifth sub-section demonstrates the effectiveness of the incorporated EDA component for search exploration.

4.0.1. *Benchmarks and experimental protocols.* In order to evaluate our proposed EDA-TS algorithm, we use have different sizes of benchmarks n = 500; 750; 1,000; 1,500; 2,000; 3,000; 5,000 with a total of 140 instances, where each size of benchmarks include 20 instances from two groups (TypeI and TypeII). These instances are widely used in the literature for performance comparisons among algorithms and can be downloaded from the websites1. Similar to other reference algorithms, we use time limits as the stopping condition, which are set to be 100 seconds, 1,000 seconds and 2,000 seconds for problem instances with n 1,000, 1,000 $< n <$5,000 and n = 5,000, respectively. Our EDA-TS algorithm is programmed in C++ and compiled using GNU g++ with O3 ag on an Intel Xeon Processor E5-2670 v2 with 2.50GHz CPU. Given the random nature of our EDA-TS algorithm, we give 20 independent runs for each instance.

Our RLTS algorithm was programmed in c++ and compiled using GNU g++. At runtime, we used the taurus2 server of the cluster platform. The CPU used by the taurus2 server is the Intel Xeon Processor E5-2670 (2.5GHz 2 $\times$ 10 core (20)), and the GPU is Nvidia Tesla K20m. The software used in later parameter tuning was IBM SPSS Statistics version 22. The instances for the max–mean dispersion problem used in our paper can be divided into two types, Types I and II. The distance between elements in Type I is randomly generated in [-10, 10]

and has uniform probability distribution. The distance between elements in Type II is randomly generated in $[-10,-5] \cup [5,10]$ and also has uniform probability distribution. Accordingly, instances can also be divided into small- and medium-sized instances, and large-scale instances. Small- and medium-sized instances refer to case $n = 500, 750, 1,000$ with a total of 60 instances in each group of Type I. Each scale has 20 instances, and each type has 10 instances. These instances are all the same as those used in [15, 18, 8, 9, 6] and can be downloaded at http://grafo.etsii.urjc.es/optsicom/edp/. Large-scale instances refer to the case with $n >$ 1,000 elements. We used 40 instances with a scale of 3,000 and 5,000 variables from Type II, each scale had 20 instances, and each type had 10 instances. They were also the same as those used by Lai and Hao [15] and can be downloaded from http://www.info.univ-angers.fr/~hao/maxmeandp.html.

4.1. **Parameter settings.** Our algorithm relies on six parameters: greedy factor ϵ, learning factor α, discount factor γ, search depth alpha, tee maximum tabu tenure T_{max}, and time limit t_{limit}. For ϵ, α, γ, we set ϵ=0.7, $\alpha = 0.5$, γ=0.5 after the debugging (Section 4.4). For alpha, we followed the literature [15] and set $\alpha = 50,000$, $T_{max}, = 120$. For t_{limit}, we introduced the algorithm on the basis of whether running time exceeded limited time to determine whether to terminate the operation, and limited time needed to be set according to the size of the instance. Given n number of elements, when $500 \leq n \leq$ 1,000, time limit $t_{limit} = 100$s, when $n = 3,000$, time limit $t_{limit} = 1,000s$, when $n = 5,000$, time limit $t_{limit} = 2,000s$; the setting of time limit t_{limit} refers to the current optimal literature [15].

4.2. **Experiment results and comparisons of small- and medium-sized instances.** We ran each test 20 times and observed both maximum objective-function value f_{best} and average objective function value f_{avg} of the 20 operation results. Table 1 shows the results for small- and medium-sized instances (20 instances for each categorized scale). For the first row, Column 1 is instance name, Column 2 is instance size, Columns 3 to 9 list the chosen state-of-the-art algorithms and our proposed RLTS algorithm. In the second row, the two subcolumns in Columns 7 to 9 stand for maximum objective-function value f_{best} (for example in the results of MAMMDP algorithm), which is the current optimal algorithm [15], and average objective-function value f_{avg} (for example, average objective function value f_{avg} in the RLTS).

According to the results in Table 1 for the small- and medium-sized instances of each algorithm, it can be seen that the maximum objective-function value of RLTS is basically greater than or equal to the algorithms in [18, 8, 9, 6] (reserving two decimal places in Columns 3–5 results in larger values). The computational results are the same as those of current optimal algorithm MAMMDP, and the average objective-function value of each instance is consistent with the maximum objective-function value, that is, the current maximum objective-function value can be obtained every time the algorithm runs, proving that the RLTS is very stable. To sum up, the performance of the RLTS was better than these algorithms in the literature [15, 18, 8, 9, 6] for small- and medium-sized instances. It had the same performance as current optimal algorithms MAMMDP and EDA, which proves its greater role in tackling the max–mean dispersion problem.

Table 1: Computational results of reinforcement-learning tabu-search (RLTS) algorithm on 60 benchmark instances.

Instance(n)	GRASP-PR[18]	HH[8]	HHP[9]	TP-TS[6]	MAMMDP[15]		EDA[16]		RLTS	
					f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
MDPI1(500)	78.61	81.25	81.28	81.28	81.2770	81.2770	81.2770	81.2770	81.2770	81.2770
MDPI2(500)	76.87	77.45	77.79	77.60	78.6102	78.6102	78.6102	78.6102	78.6102	78.6102
MDPI3(500)	75.69	75.31	76.30	75.65	76.3008	76.3008	76.3008	76.3008	76.3008	76.3008
MDPI4(500)	81.81	82.28	82.33	81.47	82.3321	82.3321	82.3321	82.3321	82.3321	82.3321
MDPI5(500)	78.57	80.01	80.08	79.92	80.3540	80.3540	80.3540	80.3540	80.3540	80.3540
MDPI6(500)	79.64	81.12	81.25	79.93	81.2486	81.2486	81.2486	81.2486	81.2486	81.2486
MDPI7(500)	75.50	78.09	78.16	77.71	78.1645	78.1645	78.1645	78.1645	78.1645	78.1645
MDPI8(500)	76.98	79.01	79.06	78.70	79.1399	79.1399	79.1399	79.1399	79.1399	79.1399
MDPI9(500)	75.72	76.98	77.36	77.15	77.4210	77.4210	77.4210	77.4210	77.4210	77.4210
MDPI10(500)	80.38	81.24	81.25	81.02	81.3099	81.3099	81.3099	81.3099	81.3099	81.3099
MDPII1(500)	108.15	109.16	109.38	109.33	109.6101	109.6101	109.6101	109.6101	109.6101	109.6101
MDPII2(500)	103.29	105.06	105.33	104.81	105.7175	105.7175	105.7175	105.7175	105.7175	105.7175
MDPII3(500)	106.30	107.64	107.79	107.18	107.8217	107.8217	107.8217	107.8217	107.8217	107.8217
MDPII4(500)	104.62	105.37	106.10	105.69	106.1001	106.1001	106.1001	106.1001	106.1001	106.1001
MDPII5(500)	103.61	106.37	106.55	106.59	106.8572	106.8572	106.8572	106.8572	106.8572	106.8572
MDPII6(500)	104.81	105.52	105.77	106.17	106.2980	106.2980	106.2980	106.2980	106.2980	106.2980
MDPII7(500)	104.50	106.61	107.06	106.92	107.1494	107.1494	107.1494	107.1494	107.1494	107.1494
MDPII8(500)	100.02	103.41	103.78	103.49	103.7792	103.7792	103.7792	103.7792	103.7792	103.7792
MDPII9(500)	104.93	106.20	106.24	105.97	106.6198	106.6198	106.6198	106.6198	106.6198	106.6198
MDPII10(500)	103.50	103.79	104.15	103.56	104.6515	104.6515	104.6515	104.6515	104.6515	104.6515
MDPI1(750)	–	–	–	95.86	96.6507	96.6507	96.6507	96.6507	96.6507	96.6507
MDPI2(750)	–	–	–	97.42	97.5649	97.5649	97.5649	97.5649	97.5649	97.5649
MDPI3(750)	–	–	–	96.97	97.7989	97.7989	97.7989	97.7989	97.7989	97.7989
MDPI4(750)	–	–	–	95.21	96.0414	96.0414	96.0414	96.0414	96.0414	96.0414
MDPI5(750)	–	–	–	96.65	96.7619	96.7619	96.7619	96.7619	96.7619	96.7619
MDPI6(750)	–	–	–	99.25	99.8613	99.8613	99.8613	99.8613	99.8613	99.8613
MDPI7(750)	–	–	–	96.26	96.5454	96.5454	96.5454	96.5454	96.5454	96.5454
MDPI8(750)	–	–	–	96.46	96.7270	96.7270	96.7270	96.7270	96.7270	96.7270
MDPI9(750)	–	–	–	96.78	98.0584	98.0584	98.0584	98.0584	98.0584	98.0584
MDPI10(750)	–	–	–	99.85	100.0642	100.0642	100.0642	100.0642	100.0642	100.0642
MDPII1(750)	–	–	–	127.69	128.8637	128.8637	128.8637	128.8637	128.8637	128.8637

MDPII2(750)	–	–	–	130.79	130.9544	130.9544	130.9544	130.9544	130.9544	130.9544
MDPII3(750)	–	–	–	129.40	129.7825	129.7825	129.7825	129.7825	129.7825	129.7825
MDPII4(750)	–	–	–	125.68	126.5823	126.5823	126.5823	126.5823	126.5823	126.5823
MDPII5(750)	–	–	–	128.13	129.1229	129.1229	129.1229	129.1229	129.1229	129.1229
MDPII6(750)	–	–	–	128.55	129.0252	129.0252	129.0252	129.0252	129.0252	129.0252
MDPII7(750)	–	–	–	124.91	125.6467	125.6467	125.6467	125.6467	125.6467	125.6467
MDPII8(750)	–	–	–	130.66	130.9405	130.9405	130.9405	130.9405	130.9405	130.9405
MDPII9(750)	–	–	–	128.89	128.8899	128.8899	128.8899	128.8899	128.8899	128.8899
MDPII10(750)	–	–	–	132.99	133.2653	133.2653	133.2653	133.2653	133.2653	133.2653
MDPI1(1,000)	–	–	–	118.76	119.1741	119.1741	119.1741	119.1741	119.1741	119.1741
MDPI2(1,000)	–	–	–	113.22	113.5248	113.5248	113.5248	113.5248	113.5248	113.5248
MDPI3(1,000)	–	–	–	114.51	115.1386	115.1386	115.1386	115.1386	115.1386	115.1386
MDPI4(1,000)	–	–	–	110.53	111.1504	111.1504	111.1504	111.1504	111.1504	111.1504
MDPI5(1,000)	–	–	–	111.24	112.7232	112.7232	112.7232	112.7232	112.7232	112.7232
MDPI6(1,000)	–	–	–	112.08	113.1987	113.1987	113.1987	113.1987	113.1987	113.1987
MDPI7(1,000)	–	–	–	110.94	111.5555	111.5555	111.5555	111.5555	111.5555	111.5555
MDPI8(1,000)	–	–	–	110.29	111.2632	111.2632	111.2632	111.2632	111.2632	111.2632
MDPI9(1,000)	–	–	–	115.78	115.9588	115.9588	115.9588	115.9588	115.9588	115.9588
MDPI10(1,000)	–	–	–	114.29	114.7316	114.7316	114.7316	114.7316	114.7316	114.7316
MDPII1(1,000)	–	–	–	145.46	147.9362	147.9362	147.9362	147.9362	147.9362	147.9362
MDPII2(1,000)	–	–	–	150.49	151.3800	151.3800	151.3800	151.3800	151.3800	151.3800
MDPII3(1,000)	–	–	–	149.36	150.7882	150.7882	150.7882	150.7882	150.7882	150.7882
MDPII4(1,000)	–	–	–	147.91	149.1780	149.1780	149.1780	149.1780	149.1780	149.1780
MDPII5(1,000)	–	–	–	150.23	151.5208	151.5208	151.5208	151.5208	151.5208	151.5208
MDPII6(1,000)	–	–	–	147.29	148.3434	148.3434	148.3434	148.3434	148.3434	148.3434
MDPII7(1,000)	–	–	–	148.41	148.7424	148.7424	148.7424	148.7424	148.7424	148.7424
MDPII8(1,000)	–	–	–	145.87	147.8268	147.8268	147.8268	147.8268	147.8268	147.8268
MDPII9(1,000)	–	–	–	145.67	147.0839	147.0839	147.0839	147.0839	147.0839	147.0839
MDPII10(1,000)	–	–	–	148.40	150.0461	150.0461	150.0461	150.0461	150.0461	150.0461

4.3. **Results and comparisons on large-scale instances.** Table 2 shows the results for large-scale instances, including 20 instances with scales of both 3,000 and 5,000. Since the algorithms in [18, 8, 9] cannot compute large-scale instances, the following table only lists the TP-TS algorithm in [6], MAMMDP algorithm in [15], EDA algorithm [16], and our RLTS algorithm. The data in bold indicate that the result was better than MAMMDP, and the underlined data indicate that the result was inferior to MAMMDP.

Furthermore, according to the results in the above Table 1, the maximum objective function value computed by our RLTS algorithm is larger than TP-TS algorithm in [6]. Compared with EDA algorithm, among these 40 instances, there are two maximum objective function values that are not as good as EDA and one is greater than EDA, however, in terms of stability, there are 19 average objective function values larger than EDA, only 6 is not as good. Thus, the stability of our proposed algorithm is stronger than EDA. Compared with MAMMDP algorithm which is the current best algorithm, the computational results when solving the problem instances with 3,000 variables are all the same; while with 5,000, the maximum objective function values of 3 instances are greater than MAMMDP, and 1 instance is smaller than MAMMDP. In terms of the average objective function values, in the 3,000-scale instances have 5 average objective function values greater than MAMMDP, against one less than. In the 5,000-scale instances, the average objective function value of 5 instances is greater than MAMMDP, and the rest are all less than. Therefore, it is illustrated that the RLTS algorithm almost has the same ability to obtain the maximum objective function value as compared with MAMMDP algorithm. In terms of stability, the RLTS is more stable when solving 3,000-scale instances while MAMMDP is more stable when solving $5,000_scale$ instances. In addition, the Table 2 shows the average computational time required by the RLTS to achieve better solutions is shorter than those both MAMMDP and EDA algorithms taken as to be better among all state-of-the-art algorithms used in this work. Although the RLTS does not get the current optimal solution in one instance, it also updates the records of three optimal solutions. This shows that the RLTS algorithm designed in this paper is still superior when tackling the max–mean dispersion problems.

Furthermore, we study components of our RLTS algorithm to analyze their influence on its performance, including significance analysis of parameters ϵ, α, γ and the role of the reinforcement learning.

4.4. **Significance analysis of parameters $\alpha, \epsilon, t, \gamma$.** Two factors α and γ mainly affect reinforcement learning when experience matrix Q is updated (Section 2.4). The greater learning factor α is, the smaller the degree to which the original value is retained each time experience matrix Q is updated, and the Q value retains the overall reward obtained by this action, that is, the update degree is larger. If learning factor α is smaller, the original value is more retained when the Q-value is updated, and it is integrated into the comprehensive reward brought by this action to a lesser extent. The larger discount factor γ is, the greater the discount rate of future rewards when the Q value is updated, that is, the greater the degree of consideration for future rewards in the action selection is. The smaller discount factor γ is, the smaller the discount rate of future rewards is when Q value is updated, that is, the current reward is more considered in the action selection, and future rewards only account for a small portion.

Table 2: Computational results of RLTS algorithm on the 40 benchmark large instances with $n = 3,000, 5,000$. Each instance was independently solved 20 times, and results were compared to the TP-TS, MAMMDP, and EDA algorithms.

Instance(n)	TP-TS	MAMMDP			EDA			RLTS		
		f_{best}	f_{avg}	$time$	f_{best}	f_{avg}	time	f_{best}	f_{avg}	$time$
MDPI1(3,000)	188.0953	189.049	189.049	88.36	189.049	189.049	69.35	189.049	189.049	47.45
MDPI2(3,000)	186.4730	187.3873	187.3873	60.71	187.3873	187.3873	53.64	187.3873	187.3873	66.65
MDPI3(3,000)	184.3414	185.6668	185.6551	352.85	185.6668	185.6453	399.44	185.6668	185.6622	426.7
MDPI4(3,000)	185.5882	186.1637	186.1536	300.37	186.1637	186.1637	240.45	186.1637	186.1637	137.8
MDPI5(3,000)	186.2349	187.5455	187.5455	61.29	187.5455	187.5455	94.16	187.5455	187.5455	54.23
MDPI6(3,000)	189.0935	189.4313	189.4313	51.99	189.4313	189.4313	48.21	189.4313	189.4313	23.15
MDPI7(3,000)	187.4512	188.2426	188.2426	86.57	188.2426	188.2426	120.95	188.2426	188.2426	65.45
MDPI8(3,000)	185.7358	186.7968	186.7968	48.04	186.7968	186.7968	38.7	186.7968	186.7968	31.2
MDPI9(3,000)	187.1076	188.2313	188.2313	151.78	188.2313	188.2313	82.66	188.2313	188.2313	47.25
MDPI10(3,000)	184.6866	185.6825	185.6238	228.72	185.6825	185.6719	510.41	185.6825	185.6822	467.1
MDPII1(3,000)	252.1818	252.3204	252.3204	59.7	252.3204	252.3204	42.42	252.3204	252.3204	51.2
MDPII2(3,000)	248.6972	250.0621	250.0621	220.1	250.0621	250.0617	248.1	250.0621	250.0576	514.4
MDPII3(3,000)	250.5303	251.9063	251.9063	146.32	251.9063	251.9063	139.36	251.9063	251.9063	78.1
MDPII4(3,000)	253.0963	253.941	253.9406	370.76	253.941	253.9406	352.17	253.941	253.941	184.05
MDPII5(3,000)	252.5621	253.2604	253.2604	374	253.2604	253.2603	308.8	253.2604	253.2608	344.15
MDPII6(3,000)	249.7160	250.6778	250.6778	55.35	250.6778	250.6778	55.9	250.6778	250.6778	37.25
MDPII7(3,000)	249.5939	251.1344	251.1344	74.72	251.1344	251.1344	88.61	251.1344	251.1344	59.85
MDPII8(3,000)	252.0565	252.9996	252.9996	79.82	252.9996	252.9996	123.56	252.9996	252.9996	61.72
MDPII9(3,000)	251.3625	252.4258	252.4258	90.27	252.4258	252.4258	58.5	252.4258	252.4258	94.4
MDPII10(3,000)	251.1169	252.3966	252.3966	13.18	252.3966	252.3966	8.73	252.3966	252.3966	12.02
MDPI1(5,000)	236.3332	4395.321	4395.24	2914.9	4395.321	4395.288	3084.12	4395.321	4395.312	2804.12
MDPI2(5,000)	239.0143	219.7661	219.762	145.745	219.7661	219.7644	154.206	219.7661	219.7656	140.206
MDPI3(5,000)	238.4742	240.1625	240.1029	312.13	240.1625	240.0434	905.58	240.1625	240.0519	1061.45
MDPI4(5,000)	237.3972	241.8274	241.793	1244.36	241.8274	241.768	958.6	241.8274	241.7649	1046.76
MDPI5(5,000)	240.0439	240.8908	240.8882	810.48	240.8908	240.8494	992.57	240.8908	240.8518	1040.85
MDPI6(5,000)	238.0015	240.9972	240.9768	653.64	240.9972	240.9281	1240.33	240.9925	240.9129	910.6

MDPI7(5,000)	239.7444	242.4801	242.4759	735.16	242.4801	242.4419	1104.64	242.4801	242.4593	895.02
MDPI8(5,000)	237.9150	240.3229	240.3063	976.02	240.376	240.2726	992.36	240.376	240.2698	1072.8
MDPI9(5,000)	235.9103	242.8149	242.775	259.5	242.8201	242.7624	965.79	242.8201	242.7778	1036.81
MDPI10(5,000)	241.8043	241.195	241.1618	1148.6	241.195	241.1291	909.39	241.195	241.134	1036.05
MDPII1(5,000)	316.7478	239.7606	239.6676	1219.71	239.7606	239.5363	1001.4	239.7606	239.6131	1441.87
MDPII2(5,000)	323.6829	243.4737	243.373	457.28	243.4737	243.3442	981.99	243.4737	243.3673	758.61
MDPII3(5,000)	321.9291	322.2359	322.1813	1519.05	322.2359	322.1594	1114.21	322.2359	322.1691	1089.2
MDPII4(5,000)	317.6767	327.3019	327.0063	1103.13	327.3019	327.1024	731.65	327.3019	327.2153	858.95
MDPII5(5,000)	317.7479	324.8135	324.8016	955.81	324.8135	324.777	1043.24	324.8135	324.7917	1039.47
MDPII6(5,000)	319.3890	322.2376	322.1973	664.1	322.2277	322.1171	1059.47	322.2376	322.1448	1106.05
MDPII7(5,000)	319.9806	322.4912	322.3807	1014.9	322.5012	322.3717	971.06	322.5012	322.3647	1151.2
MDPII8(5,000)	318.8545	322.9505	322.7039	352.88	322.9505	322.6878	1405.72	322.9505	322.7466	937.45
MDPII9(5,000)	320.4376	322.8504	322.7931	714.31	322.8504	322.7615	1164.19	322.8504	322.8136	1052.93
MDPII10(5,000)	320.8530	323.1121	323.0533	879.48	323.1121	322.8815	1168.86	323.1121	322.8943	1015.75

Table 3: Parameters of ϵ, α, and γ

Names	Range	Debugging Intervals	Final values
Greedy factor	[0.5,1)	0.05	0.7
Learning factor	(0,1)	0.1	0.5
Discount factor	[0,1)	0.1	0.5

Table 4: Debugging results for factor ϵ.

Instances (n) / ϵ	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
MDPI1(5,000)	-0.122851	-0.081087	-0.130172	-0.128802	-0.056058	-0.129919	-0.041805	-0.155847	-0.044694	-0.105110
MDPI2(5,000)	-0.041683	-0.044909	-0.042706	-0.031403	-0.032596	-0.058119	-0.012692	-0.060083	-0.056580	-0.054879
MDPI3(5,000)	-0.067597	-0.050908	-0.053767	-0.106229	-0.068687	-0.076957	-0.105697	-0.085361	-0.115044	-0.051265
MDPI4(5,000)	-0.133373	-0.042282	-0.115774	-0.091527	-0.064438	-0.134837	-0.063109	-0.100193	-0.061781	-0.071073
MDPI5(5,000)	-0.042153	-0.058244	-0.073869	-0.027768	-0.041856	-0.064060	-0.072030	-0.051006	-0.032583	-0.059985
MDPI6(5,000)	-0.040458	-0.030610	-0.054468	-0.079938	-0.042002	-0.093500	-0.018872	-0.071949	-0.053420	-0.030553
MDPI7(5,000)	-0.007039	-0.040785	-0.022835	-0.025523	-0.011432	-0.022520	-0.000364	-0.015116	0.004182	-0.010404
MDPI8(5,000)	-0.048888	-0.064965	-0.040440	-0.050873	-0.058277	-0.058174	-0.040082	-0.054745	-0.050569	-0.078426
MDPI9(5,000)	-0.129854	-0.078371	-0.117872	-0.097018	-0.076916	-0.176947	-0.060925	-0.175825	-0.084588	-0.097384
MDPI10(5,000)	-0.033717	-0.043167	-0.010259	-0.039291	-0.035665	-0.062412	-0.042826	-0.063820	-0.018992	-0.030091
MDPII1(5,000)	-0.005740	-0.033777	-0.059031	-0.047182	-0.044300	-0.066763	-0.060716	-0.031978	-0.014357	-0.039396
MDPII2(5,000)	0.147624	0.167856	0.178805	0.132264	0.164916	0.118441	0.083772	0.211362	0.147558	0.088298
MDPII3(5,000)	-0.023621	-0.023856	-0.045420	-0.022302	-0.009195	-0.030358	-0.013416	-0.023812	-0.027812	-0.027001
MDPII4(5,000)	-0.166333	-0.094202	-0.120204	-0.096423	-0.085864	-0.121264	-0.120666	-0.188106	-0.240823	-0.147536
MDPII5(5,000)	-0.075976	-0.096680	-0.044135	-0.004540	-0.033960	-0.043347	-0.077273	-0.032238	-0.036416	-0.031091
MDPII6(5,000)	-0.017563	0.058637	0.005761	-0.098428	-0.080234	0.015629	-0.008452	0.021738	0.033528	-0.012549
MDPII7(5,000)	-0.015845	-0.074944	-0.060637	-0.081458	-0.021804	-0.056874	-0.053205	-0.017029	-0.017963	-0.067988
MDPII8(5,000)	-0.178216	-0.143775	-0.177740	-0.229257	-0.176900	-0.163983	-0.205498	-0.172191	-0.244947	-0.188872
MDPII9(5,000)	0.014833	-0.094614	-0.111384	-0.118775	0.068232	-0.047735	-0.119738	-0.069007	-0.124918	-0.130199
MDPII10(5,000)	0.017633	-0.048853	-0.107008	-0.162074	0.012272	-0.128291	0.050908	0.021277	0.044312	-0.082684

Debugging results for factors α and γ were omitted.

Next, we show the method for obtaining debugging data, as follows: when other parameters remain unchanged, the current debugging parameter is changed according to the debugging interval. Because the calculation of large-scale instances more easily reflects differences after a parameter is changed, 20 instances with a scale of 5,000 are estimated. Each parameter value is measured 20 times, and the average objective-function value is taken. Since the objective-function values of various instances greatly differ, the mean value of the objective function is subtracted from the calculated mean value of each instance in the literature [15], and the difference is used as the debugging data. Figures 1, 2, and 3 below are the debugging results for factors ϵ, α, and γ, respectively. Table 4 shows the debugging results for factor ϵ. When the below data were analyzed, we used the Friedman test. The Friedman test is a nonparametric test method that uses rank to outline significant differences in multiple population distributions. The original assumption is that there is no significant difference in multiple population distributions from multiple paired samples. We used IBM SPSS Statistics 22 software to carry out Friedman's test on the above data and calculate corresponding probability p-value. If the p-value was less than the given significance level of 0.05, the original assumption was rejected and it was to assumed that there was significant difference in the sample ranks. In the case of the contrary, the original assumption could not be rejected, and it could be considered that there was no significant difference in the sample ranks in each group. By using Friedman's test, we could debug the most suitable values of greedy factor ϵ, learning factor α, and discount factor γ. At the same time, it could also test whether these three parameters are significant, that is, the change of these three parameters had greater impact on our algorithm. Table 5 shows the Friedman test results of these three parameters.

Table 5: Friedman test results for three parameters.

ϵ	Mean Rank	α	Mean Rank	γ	Mean Rank
ϵ 0.5	6.2	α 0.1	3.8	γ 0	4.2
ϵ 0.55	6.1	α 0.2	5.05	γ 0.1	4.15
ϵ 0.6	4.9	α 0.3	4.6	γ 0.2	4.7
ϵ 0.65	4.7	α 0.4	4.25	γ 0.3	5.6
ϵ 0.7	6.9	α 0.5	6.45	γ 0.4	5.4
ϵ 0.75	3.65	α 0.6	4.95	γ 0.5	7.15
ϵ 0.8	6.15	α 0.7	4.85	γ 0.6	5.75
ϵ 0.85	5.3	α 0.8	5.95	γ 0.7	5.85
ϵ 0.9	6.1	α 0.9	5.1	γ 0.8	6.3
ϵ 0.95	5			γ 0.9	5.9
N	20	N	20	N	20
Chi-Square	18.12	Chi-Square	13.88	Chi-Square	17.193
df	9	df	8	df	9
Asymp. Sig	0.034	Asymp. Sig.	0.085	Asymp. Sig.	0.46

In addition to the Friedman test, the box diagrams of the debug variables for each group were also displayed in SPSS to intuitively display the degree of change in the values of the parameters. The circle and star are abnormal values, and the others are, from top to bottom, upper edge, upper quartile, median, lower quartile, and lower edge. Figures 1, 2, and 3 below are the box diagrams of the debugging data for greedy factor ϵ, learning factor α, and discount factor γ.

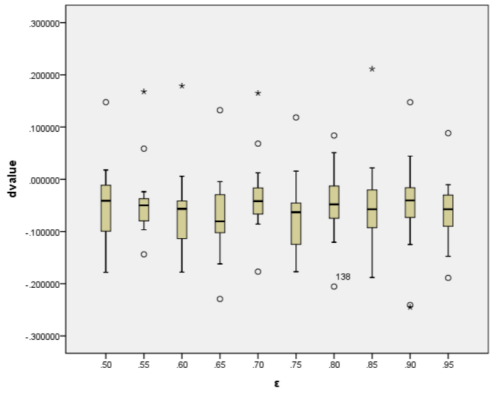

FIGURE 1. Box diagrams of debugging data for greedy factor ϵ.

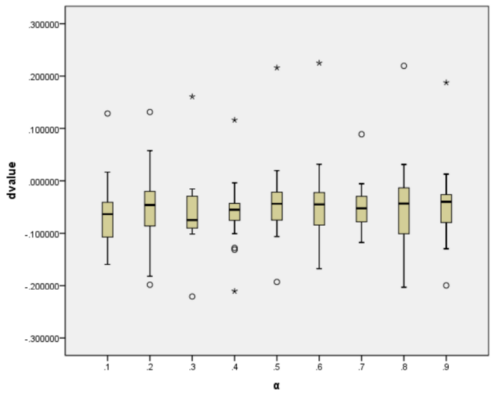

FIGURE 2. Box diagrams of debugging data for learning factor α.

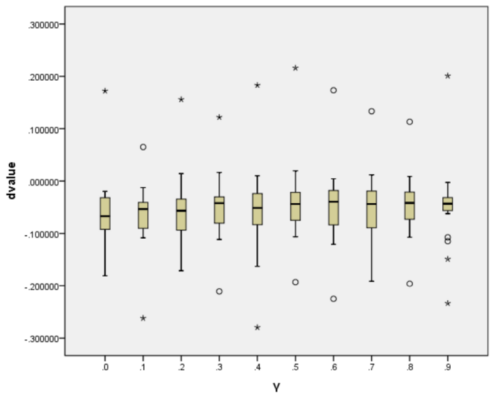

FIGURE 3. Box diagrams of debugging data for discount factor rate γ.

According to results of the Friedman test and the box diagrams of the corresponding data, it can be seen that the change of greedy factor ϵ had greater influence on the algorithm results. Probability p-value was 0.034 less than 0.05, indicating that the parameter rejected the original assumption, which is significant. However, the change of learning factor α had less influence on the algorithm. Its probability p-value was 0.085 greater than 0.05, indicating that the parameter was not significant. The change of discount factor γ affected the algorithm, but it was not obvious. Its probability p-value was 0.046 less than 0.05, but it was very close to 0.05, indicating that the parameter was significant, but that significance was not as good as that of greedy factor ϵ.

Therefore, learning factor α is a noncritical parameter in the algorithm, while greedy factor ϵ and discount factor γ are key parameters. From them, greedy factor ϵ is the most critical and has the greatest influence on the algorithm.

4.5. **Role of reinforcement learning.** In this section, the effect of the reinforcement-learning algorithm is analyzed. The specific analysis process was to compare the performance of the RLTS algorithm with the multistart tabu-search method MTS algorithm that removes the part of the reinforcement-learning algorithm, and then analyzes the effect of the reinforcement-learning algorithm on the basis of the results. Test instances of 40 large-scale instances with scales of 3,000 and 5,000 were only considered since, for small- and medium-sized instances, the results were basically the same. Each instance was tested 20 times by the two algorithms, and the final comparison was made on the basis of the maximum and average objective-function values.

Table 6 shows the results of multistart tabu-search (MTS) algorithm and the proposed RLTS algorithm for large-scale instances. The larger result of the two algorithms is in bold in the table below. From this table, it can be seen that the maximum objective-function values of the two algorithms were almost the same when considering the 3,000-scale instances, and the average objective-function value of RLTS was larger than that of MTS in five instances, which shows that the RLTS was more stable when calculating the 3,000-scale instances. While considering 5,000-scale instances, the RLTS outperformed the MTS in terms of both the maximum and the average objective-function values (greater in 11 and 16 instances than MTS, respectively). Therefore, its performance result was better than that of MTS since it could not only find a better-quality solution, it was also more stable, demonstrating the effectiveness of the reinforcement-learning algorithm.

4.6. **Effectiveness of RL component for search exploration.** In this section, a little bit different from the Section 4.2, we again demonstrated the importance of the RLTS component in the exploration of the solution space. For this, we produced two RLTS variants, using popular exploration mechanism in the literature to replace RLTS. The first is called randomized greedy tabu search (RG-TS), in which the solution is gradually built by a restricted candidate set of attributes that generates a high objective gain; then, an attribute is selected from the created set to add to the partial solution. This procedure continued until no objective gain was positive. The other variant was a randomized perturbation tabu search (RP-TS), in which the best result in the set was perturbed; i.e., a given number of attributes were modified randomly to generate a new result. Since we observed a significant decrease in performance when solving large instances compared with small- and medium-sized instances, we conducted our experiments on 20 instances with 3,000

TABLE 6. Comparison between multistart tabu-search (MTS) method and the proposed RLTS algorithm on the set of 40 large instances within $\geq$ 3,000. Each instance was independently solved 20 times by the two algorithms.

Instance (n)	MTS		RLTS	
	f_{best}	f_{avg}	f_{best}	f_{avg}
MDPI1(3,000)	189.04897	189.04897	189.04897	189.04897
MDPI2(3,000)	187.38729	187.38729	187.38729	187.38729
MDPI3(3,000)	185.66681	185.65159	185.66681	**185.6594**
MDPI4(3,000)	186.16373	186.16373	186.16373	186.16373
MDPI5(3,000)	187.54552	187.54552	187.54552	187.54552
MDPI6(3,000)	189.43126	189.43126	189.43126	189.43126
MDPI7(3,000)	188.24258	188.24258	188.24258	188.24258
MDPI8(3,000)	186.79681	186.79681	186.79681	186.79681
MDPI9(3,000)	188.23126	188.23126	188.23126	188.23126
MDPI10(3,000)	185.68251	185.67237	185.68251	**185.6747**
MDPII1(3,000)	252.32043	252.32043	252.32043	252.32043
MDPII2(3,000)	250.06214	250.05474	250.06214	**250.0569**
MDPII3(3,000)	251.90627	251.90627	251.90627	251.90627
MDPII4(3,000)	253.94101	253.93968	253.94101	**253.941**
MDPII5(3,000)	253.26042	253.26016	253.26042	**253.2604**
MDPII6(3,000)	250.67775	250.67775	250.67775	250.67775
MDPII7(3,000)	251.13441	251.13441	251.13441	251.13441
MDPII8(3,000)	252.99965	252.99965	252.99965	252.99965
MDPII9(3,000)	252.42577	252.42577	252.42577	252.42577
MDPII10(3,000)	252.39659	252.39659	252.39659	252.39659
MDPI1(5,000)	240.14121	**240.0212**	**240.1594**	240.01588
MDPI2(5,000)	241.81754	241.75355	**241.8274**	**241.7716**
MDPI3(5,000)	240.89082	240.82517	240.89082	**240.8443**
MDPI4(5,000)	240.97349	240.91546	**240.9972**	**240.9249**
MDPI5(5,000)	242.48013	242.43047	242.48013	**242.4512**
MDPI6(5,000)	240.32868	**240.2663**	240.32868	240.26432
MDPI7(5,000)	242.82014	242.7599	242.82014	**242.7793**
MDPI8(5,000)	241.14478	241.11345	**241.195**	**241.1323**
MDPI9(5,000)	239.76056	239.51496	239.76056	**239.5929**
MDPI10(5,000)	243.38549	243.34815	**243.4737**	**243.3598**
MDPII1(5,000)	322.22322	322.1312	**322.2359**	**322.1659**
MDPII2(5,000)	327.30191	327.07525	327.30191	**327.2223**
MDPII3(5,000)	324.81083	324.79022	**324.8135**	**324.7931**
MDPII4(5,000)	322.21229	**322.1266**	**322.2376**	322.08809
MDPII5(5,000)	322.42081	322.30125	**322.5012**	**322.3782**
MDPII6(5,000)	322.95049	322.61523	322.95049	**322.7672**
MDPII7(5,000)	322.85044	**322.7784**	322.85044	322.7757
MDPII8(5,000)	323.03384	322.87316	**323.1121**	**322.8885**
MDPII9(5,000)	323.52271	323.27856	**323.5438**	**323.4081**
MDPII10(5,000)	324.51991	324.29479	324.51991	**324.5191**

and 5,000 variables for the two variants of RLTS using the same settings as those of the previous tests with RLTS.We recorded the percent deviations of both the best and average values achieved by RG-TS and RP-TS from the solutions found by RLTS. The experimental results are plotted in Fig. 4.

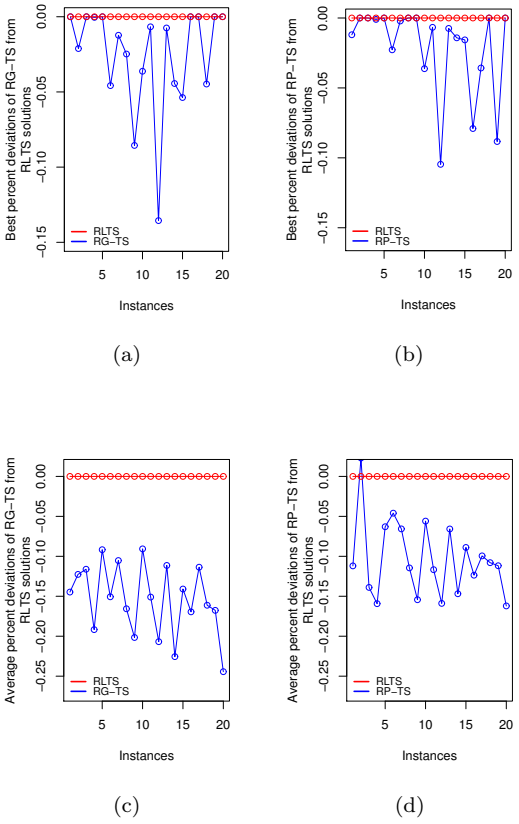

FIGURE 4. Effectiveness of the incorporated RLTS component in the proposed algorithm

The Fig. 4(a) and Fig. 4(b) showed the best percent deviations of RG-TS and RP-TS, respectively, from RLTS for each of the 20 selected instances. The Fig. 4(a) showed that the best percent deviations were non-positive for all the selected instances,while Fig. 4(b) showed that the best percent deviations of RP-TS from RLTS were non-positive for all tested instances. These results revealed that RLTS found best solution values that were better than or equivalent to those found by its two variants. The Fig. 4(c) and Fig. 4(d) showed the average percent deviations of RG-TS and RP-TS, respectively, from RLTS for each of the 20 selected instances. The Fig. 4(c) showed that the average percent deviations were negative for all the selected instances, while Fig. 4(d) showed that the average percent deviations of RP-TS from RLTS are negative for all 20 selected instances, except for one instance. These results revealed that RLTS found better average objective values than its variants. Thus, this study proved that incorporating the RL component is beneficial to the effectiveness of the algorithm

5. **Conclusion.** In this paper, we proposed for the first time strong hybrid meta-heuristics that integrates reinforcement-learning and tabu-search algorithms (RLTS)

for tackling the max–mean dispersion problem. The innovative key feature in our proposed algorithm is the use of an RL algorithm-guided solution-building strategy for diversifying the search, and a tabu-search improvement mechanism for intensifying the search. At each step, the algorithm relies on gradually setting the reward mechanism on the basis of the contrast information between initial solution and processed local optimal solution, to encourage the selection of more excellent elements when constructing the initial solution. Extensive experiments and the final computational results demonstrated that our RLTS algorithm could compete with MAMMDP, the current optimal algorithm when calculating small- and medium-scaled instances that are below 3,000 scales, and could find the current optimal solution. Performance was better than the algorithms in [18, 8, 9, 6]. When processing large-scale instances of 3,000–5,000 scales, the RLTS algorithm could also find the current optimal solution, and its stability was higher than that of MAMMDP. Moreover, our RLTS algorithm successfully updated the current optimal solutions of three instances, which proved its superiority to other algorithms for solving the max–mean dispersion problem. In addition, we analyzed parameter sensitivity and found the great role and influence of the RL algorithm component in the better performance of the outperforming RLTS algorithm. Our findings could inspire investigating other typical hybrids that integrate learning strategies and local-search mechanisms for solving NP-hard optimization problems in terms of stability improvement. Specific ideas can be attempted, such as designing more appropriate challenges to the three components, including a reward mechanism, action-selection strategy, and matrix Q updating. The RLTS algorithm can also attempt to solve other types of NP-hard dispersion or even nondispersion problems, such that those developed in literature [18, 8, 9, 6] or, for knapsack problem using hybrid based metaheuristic, in terms of the optimization mechanism.

Funding. This work was supported by the National Natural Science Foundation of China [grant number 71971170].

REFERENCES

[1] R. Aringhieri, R. Cordone and A. Grosso, Construction and improvement algorithms for dispersion problems, *European J. Oper. Res.*, **242** (2015), 21–33.
[2] R. Aringhieri and R. Cordone, Comparing local search metaheuristics for the maximum diversity problem, *J. Oper. Res. Soc.*, **62** (2011), 266–280.
[3] J. Boyan and A. W. Moore, Learning evaluation functions to improve optimization by local search, *J. Machine Learning Research*, **1** (2000), 77–112.
[4] J. Brimberg, N. Mladenović, R. Todosijević and D. Urošević, Less is more: Solving the max-mean diversity problem with variable neighborhood search, *Information Sciences*, **382** (2017), 179–200.
[5] E. K. Burke, G. Kendall and E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, *J. Heuristics*, **9** (2003), 451–470.
[6] R. Carrasco, A. Pham, M. Gallego, F. Gortázar, R. Martí and A. Duarte, Tabu search for the Max–Mean Dispersion Problem, *Knowledge-Based Systems*, **85** (2015), 256–264.
[7] F. C. De Lima Júnior, A. D. D. Neto and J. D. De Melo, Hybrid metaheuristics using reinforcement learning applied to salesman traveling problem, in *Traveling Salesman Problem, Theory and Applications*, IntechOpen, 2010.
[8] F. Della Croce, M. Garraffa and F. Salassa, A hybrid heuristic approach based on a quadratic knapsack formulation for the max-mean dispersion problem, in *Combinatorial Optimization*, Lecture Notes in Comput. Sci., 8596, Springer, Cham, 2014, 186–194.
[9] F. Della Croce, M. Garraffa and F. Salassa, A hybrid three-phase approach for the max-mean dispersion problem, *Comput. Oper. Res.*, **71** (2016), 16–22.

[10] F. Della Croce, A. Grosso and M. Locatelli, A heuristic approach for the max-min diversity problem based on max-clique, *Comput. Oper. Res.*, **36** (2009), 2429–2433.
[11] P. Galinier, Z. Boujbel and M. Coutinho Fernandes, An efficient memetic algorithm for the graph partitioning problem, *Ann. Oper. Res.*, **191** (2011), 1–22.
[12] M. Garraffa, F. Della Croce and F. Salassa, An exact semidefinite programming approach for the max-mean dispersion problem, *J. Comb. Optim.*, **34** (2017), 71–93.
[13] A. Gosavi, Reinforcement learning: A tutorial survey and recent advances, *INFORMS J. Comput.*, **21** (2009), 178–192.
[14] X. Lai, D. Yue, J.-K. Hao and F. Glover, Solution-based tabu search for the maximum min-sum dispersion problem, *Inform. Sci.*, **441** (2018), 79–94.
[15] X. Lai and J. K. Hao, A tabu search based memetic algorithm for the max-mean dispersion problem, *Comput. Oper. Res.*, **72** (2016), 118–127.
[16] P. Larranaga, A review on estimation of distribution algorithms, in *Estimation of Distribution Algorithmn*, Genetic Algorithms and Evolutionary Computation, 2, Springer, Boston, 2002, 57–100.
[17] Z. Lu, F. Glover and J.-K. Hao, Neighborhood combination for unconstrained binary quadratic programming, MIC 2009: The VIII Metaheuristics International Conference, Hamburg, Germany, 2009.
[18] R. Martí and F. Sandoya, GRASP and path relinking for the equitable dispersion problem, *Comput. Oper. Res.*, **40** (2013), 3091–3099.
[19] V. V. Miagkikh and W. F. Punch, Global search in combinatorial optimization using reinforcement learning algorithms, *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, Washington, DC, 1999.
[20] D. Nijimbere, S. Zhao, H. Liu, B. Peng and A. Zhang, A hybrid metaheuristic of integrating estimation of distribution algorithm with tabu search for the max-mean dispersion problem, *Math. Probl. Eng.*, **2019** (2019), 16pp.
[21] D. C. Porumbel, J.-K. Hao and F. Glover, A simple and effective algorithm for the MaxMin diversity problem, *Ann. Oper. Res.*, **186** (2011), 275–293.
[22] O. A. Prokopyev, N. Kong, and D. L. Martinez-Torres, The equitable dispersion problem, *European J. Oper. Res.*, **197** (2009), 59–67.
[23] A. P. Punnen, S. Taghipour, D. Karapetyan and B. Bhattacharyya, The quadratic balanced optimization problem, *Discrete Optim.*, **12** (2014), 47–60.
[24] I. Sghir, J. K. Hao, I. B. Jaafar and K. Ghédira, A multi-agent based optimization method applied to the quadratic assignment problem, *Expert Systems Appl.*, **42** (2015), 9252–9262.
[25] J. A. Torkestani and M. R. Meybodi, A cellular learning automata-based algorithm for solving the vertex coloring problem, *Expert Systems Appl.*, **38** (2011), 9237–9247.
[26] Y. Wang, Q. Wu and F. Glover, Effective metaheuristic algorithms for the minimum differential dispersion problem, *European J. Oper. Res.*, **258** (2017), 829–843.
[27] Y. Wang, J.-K. Hao, F. Glover and Z. Lü, A tabu search based memetic algorithm for the maximum diversity problem, *Engineering Appl. Artificial Intell.*, **27** (2014), 103–114.
[28] Y. Xu, D. Stern and H. Samulowitz, Learning adaptation to solve constraint satisfaction problems. Available from: https://www.microsoft.com/en-us/research/wp-content/uploads/2009/01/lion2009.pdf.
[29] T. Yu and W.-G. Zhen, A multi-step $Q(\lambda)$ learning approach to power system stabilizer, *IFAC Proceedings Volumes*, **43** (2010), 220–224.
[30] Y. Zhou, J.-K. Hao, and B. Duval, Reinforcement learning based local search for grouping problems: A case study on graph coloring, *Expert Systems Appl.*, **64** (2016), 412–422.

Received May 2019; revised December 2019.

E-mail address: nijd@mail.nwpu.edu.cn
E-mail address: zhaosongzheng@nwpu.edu.cn
E-mail address: guxh@mail.npwu.edu.cn
E-mail address: moses@nwpu.edu.cn
E-mail address: nyiribakwedom@gmail.com