Penalty Variable Neighborhood Search for the Bounded Single-Depot Multiple Traveling Repairmen Problem

Ha-Bang Ban
School of Information and Communication Technology
Hanoi University of Science and Technology, Hanoi, Vietnam
E-mail: BangBH@soict.hust.edu.vn

Keywords: nounded-mTRP, penalty variable neighborhood search, metaheuristic

Received: June 4, 2019

Multiple Traveling Repairmen Problem (mTRP) is a class of NP-hard combinatorial optimization problems with many practical applications. In this paper, a general variant of mTRP, also known as the Bounded Single-Depot Multiple Traveling Repairmen Problem (Bounded-mTRP), is introduced. In the Bounded-mTRP problem, a fleet of identical vehicles is dispatched to serve a set of customers. Each vehicle that starts from the depot is only allowed to visit the number of customers within a predetermined interval, and each customer must be visited exactly once. Such restrictions appear in real-life applications where the purpose is to have a good balance of workloads for the repairmen. The goal is to find the order of customer visits that minimizes the sum of waiting times. In our work, the proposed algorithm is encouraged by the efficiency of the algorithms in [15, 19, 20] that are mainly based on the principles of the VNS [14]. The penalty VNS extends the well-known VNS [14] by including constraint penalization, to solve the Bounded-mTRP effectively. Extensive numerical experiments on benchmark instances show that our algorithm reaches the optimal solutions for the problem with 76 vertices at a reasonable amount of time. Moreover, the new best-known solutions are found in comparison with the state-of-the-art metaheuristic algorithms.

Povzetek: Razvita je nova metoda za preiskovanje grafov - za reševanje naloge serviserja, tipičnega NP-polnega problema.

1 Introduction

1.1 Motivation and definition

The Traveling Repairman Problem (TRP) has been studied in the number of previous work [1, 2, 5, 15, 19, 20]. It is known as the Minimum Latency Problem (MLP), or the Deliveryman Problem (DMP). These problems arise applications, e.g., whenever repairmen or servers have to accommodate a set of requests to minimize their total (or average) waiting times [1, 2, 5, 15, 19, 20]. A direct generalization of the TRP is the Multiple Traveling Repairmen Problem (mTRP) that considers k vehicles simultaneously. Applications of the mTRP can be found in Routing Pizza Deliverymen, or Scheduling Machines to minimize mean flow time for jobs. Several prior studies that we can find in the literature are [10, 12]. In this paper, we study the Bounded Single-Depot Multiple Traveling Repairmen Problem (Bounded-mTRP) by involving the restriction of the number of vertices that a repairman must visit in his tour. The restriction is defined by lower (denoted by K) and upper (denoted by L) bounds regarding the traveled vertices. Therefore, the number of vertices that a repairman can visit lies within a predetermined interval with the aim of obtaining balanced solutions. The requirement of the problem is to find a tour such that the above restriction is satisfied, and the overall cost of visiting all vertices is minimized. Such restriction appears in many real-life applications whose purpose is to have a good balance of workloads for the repairmen.

1.2 Approach and contributions

There are three approaches for solving the Bounded-mTRP: 1) exact algorithms, 2) approximation algorithms, and 3) heuristic algorithms. The exact algorithms find the optimal solution with an exponential time in the worst case. Therefore, the exact algorithm only solves the problem with small sizes. To describe related works, we denote an approximation algorithm as p-approximation when the algorithm finds the solution at most p times worse than the optimal solution. Here p is an approximation ratio with a constant value. In this approach, the best approximation ratio of 16.994 is for the mTRP [10, 12]; however, it is still far from the optimal solution. Heuristic algorithms perform well in practice, and their efficiency can be evaluated through experiments. Our algorithm falls into this approach.

Previously, research on the Bounded-mTRP has not studied much, and this work presents the first metaheuristic approach for this problem. Our algorithm is encouraged by the efficiency of the algorithms in [14, 19, 20] that are mainly based on the principles of the VNS [14]. However,

the difference between the Penalty VNS (P-VNS) and their VNS is that our algorithm builds up penalty value during a search. The proposed algorithm includes two phases. The algorithm is developed based on the GRASP [9] to build an initial solution in the construction phase. In the improvement phase, the P-VNS combined with shaking techniques not only exploits good local solution space but also prevents the search from escaping from local optimal. Moreover, several novel neighborhoods' structure as well as a constant time operation for calculating the cost of each neighboring solution is also introduced. The main problem is that there exists no other metaheuristic reported in the literature for this problem; this is, we found no previous attempts to solve this problem, neither exact nor heuristically, to compare with. Therefore, we adapt the metaheuristic algorithms in [17] to solve the Bounded-mTRP, and choose several state-of-the-art metaheuristic algorithms for the mTRP [8, 18], and Bounded-mTSP [17] as a baseline in our research. Extensive numerical experiments on benchmark instances show that our algorithm reaches the optimal solutions for the problems with up to 76 vertices at a reasonable amount of time. Moreover, the new best-known solutions are found in comparison with the state-of-the-art metaheuristic algorithms.

The rest of this paper is organized as follows. Section 2, and 3 present literature review, and neighborhood structure, respectively. The proposed algorithm is described in Section 4. Computational evaluations and discussions are reported in Section 5. Finally, Section 7 concludes the paper.

2 Literature review

The Bounded-mTRP has, as we know, not been studied much, although it is a natural extension of the mTRP problem. In the literature, several variants of the problem are introduced as follows:

- The mTRP is a popular case since no constraint is considered. Numerous works for the mTRP can be found in [8, 10, 12, 18]. Some metaheuristic algorithms [10, 12] can give good solutions fast for large instances.
- The mTRP with Profits (mTRPP) finds a travel plan for server that maximizes the total revenue. Metaheuristic algorithm [3] produces solutions well.
- Another variant of the mMLP is mMLP with distance constraints [4, 16]. Lou et al. [16] proposed an exact algorithm that reaches the optimal solutions for the instances with up to 50 vertices. Ban et al. [4] then presented a metaheuristic algorithm based on VNS. The experimental results concluded that the algorithm found good-quality solutions for small and medium-size instances.
- mTRPD [6] finds a tour with minimum latency sum in post-disaster road clearance. Unlike mMLP, in disaster situations, travel costs need to be added to debris removal times. Their metaheuristic obtained the optimal or near-optimal solutions on Istanbul data within seconds.
- The TRP is a particular case where there is only a repairman. Numerous metaheuristic algorithms [1, 2, 5, 15, 19, 20] for the problem have proposed in the literature. The experimental results showed that their algorithms obtain good solutions fast for the instances with up to 500 vertices.

These algorithms are the best algorithms for some variants of the Bounded-mTRP problem. However, they do not involve the bounded constraint. Therefore, they cannot be used directly to solve the Bounded-mTRP.

3 Mathematical formulation

The formulation is obtained from the formulation proposed by Christofides et al. [7] for the Capacitated Vehicle Routing Problem (CVRP). Let u_i be a non-negative real variable representing the length of the route from the depot 0 to the vertex *i*. Let x_{ij}^r be the following binary variables:

$$x_{ij}^r = \begin{cases} 1 & \text{if if edge}(i,j) \text{ is used in route } r \\ 0 & \text{otherwise} \end{cases} \quad \min z = \sum_{i=1}^{n} u_i$$

Subject to:

$$\sum_{\substack{i=0 \\ i\neq j}}^{n} \sum_{r=1}^{m} x_{ij}^r = 1; \qquad (j = 1, 2, ..., n) \tag{1}$$

$$\sum_{\substack{i=0 \\ i\neq l}}^{n} x_{il}^r - \sum_{\substack{j=0 \\ j\neq l}}^{n} x_{lj}^r = 0; \qquad (l = 0, 1, ..., n; \; r = 1, 2, ..., m) \tag{2}$$

$$\sum_{j=1}^{n} x_{0j}^r = 1; \qquad (r = 1, 2, ..., m) \tag{3}$$

$$K \leq \sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij}^r + 1 \leq L; \qquad (r = 1, 2, ..., m) \tag{4}$$

$$u_i - u_j + (T + c_{ij}) \times \sum_{r=1}^{m} x_{ij}^r + (T - c_{ji}) \times \sum_{r=1}^{m} x_{ji}^r \leq T; \quad (i, j = 1, ..., n; i \neq j) \tag{5}$$

$$u_i \geq c_{0i} \times \sum_{r=1}^{m} x_{0i}^r; \qquad (i = 1, 2, ..., n) \tag{6}$$

$$x_{ij}^r \in \{0, 1\}; \qquad (i, j = 0, 1, ..., n; j \neq i; \quad r = 1, ..., m) \tag{7}$$

$$u_i \geq 0; \qquad (i = 1, ..., n) \tag{8}$$

Constraints (1) show that each vertex is contained in only one route. Constraints (2) indicate that when a vertex is in a route, then it has a predecessor and a successor in that route. Constraints (3) ensure that one vertex is sequenced as first in each route and constraints (4) guarantee that and the number of vertices visited of each repairman must less than *L* and more than *K*. Constraints (5) demonstrate that $u_j = u_i + c_{ij}$ when $\sum_{r=1}^{m} x_{ji}^r = 1$ and they are redundant when $\sum_{r=1}^{m} x_{ji}^r = 0$. Constraints (6) initialize the latency of the first vertex in each route. Finally, constraints (7) and (8) establish the nature of the variables.

4 Neighborhood structure

Seven neighborhoods investigated are divided into two categories: intro-route and intra-route. Now, let $T = (R_1, R_2, ..., R_l, ..., R_k)$ $(l = 1, ..., k)$ be a tour, we introduce a novel neighborhoods' structure and complexity of their exploration. Note that: in [15, 20], the complexity of some neighborhoods is already mentioned. Therefore, we only introduce the complexity of new ones.

For Intro-route: Intro-route is used to optimize on a single route. Assume that, R and m $(m < n)$ are a route and its length, respectively. We then introduce five neighborhoods' structure in turn.

Remove-insert neighborhood considers each vertex v_i in the route at the end of it. This neighborhood of R is defined as a set $N_1(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_{i+1}, ..., v_m, v_i) : i = 2, 3, ..., m-1\}$. Obviously, the size of $N_1(R)$ is $O(m)$.

Property 1. The time complexity of exploring $N_1(R)$ is $O(m^2)$.

Swap adjacent neighborhood attempts to swap each pair of adjacent vertices in the route. This neighborhood of R is defined as a set $N_2(R) = \{R_i = (v_1, v_2, ..., v_{i-2}, v_i, v_{i-1}, v_{i+1}, ..., v_m) : i = 3, 4, ..., m-1\}$. The size of the neighborhood is $O(m)$.

Property 2. The time complexity of exploring $N_2(R)$ is $O(m)$.

Swap neighborhood attempts to swap the positions of each pair of vertices in the route. This neighborhood of R is defined as a set $N_3(R) = \{R_{ij} = (v_1, v_2, ..., v_{i-1}, v_j, v_{i+1}, ..., v_{j-1}, v_i, v_{j+1}, ..., v_m) : i = 2, 3, ..., m-3; j = i+3, ..., m\}$. The size of the neighborhood is $O(m^2)$.

Property 3. The complexity of exploring $N_3(R)$ is $O(m^2)$.

3-opt neighborhood attempts to reallocate three adjacent vertices to another position of the route. This neighborhood of R is defined as a set $N_4(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_i, v_{j+1}, ..., v_k, v_{i+1}, ..., v_j, v_{k+1}...., v_m) : i = 2, 3, ..., m-5, j = 4, ..., m-3, k = 6, ..., m-1\}$. The size of the neighborhood is $O(m^3)$.

Property 4. The complexity of exploring $N_4(R)$ is $O(m^3)$.

2-opt neighborhood removes each pair of edges from the solution and reconnects the vertices. This neighborhood of T is defined as a set $N_5(T) = \{T_{ij} = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+2}, v_{i+1}, v_{j+1}, ..., v_m) : i = 1, ..., n-4; j = i+4, ..., m\}$. The size of the neighborhood is $O(m^2)$.

Property 5. The complexity of exploring $N_5(T)$ is $O(m^2)$.

It is realized that the calculation of a neighboring solution's cost by using the known cost of the current solution can be done in constant time [15, 20]. As a result, the algorithm spends $O(m^3)$ operations for a full neighborhood search.

For intra-route: Let R_l, R_h, ml, and mh be two different routes and their sizes in T, respectively. Intra-route is used to exchange vertices between two different routes or remove vertices from a route and then insert them to another as followings:

The swap-intra-routes neighborhood tries to exchange the positions of each pair of vertices in R_l and R_h in turn. The neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, ..., R_2, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{il}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-1, ih = 2, 3, ..., mh-1\}$. The size of the neighborhood is $O(ml \times mh)$.

Property 6. The complexity of exploring $N_6(T)$ is $O(ml \times mh)$.

Proof. For a tour $T \in N_6(R)$, we have

$$\begin{aligned} L(T_i) \quad = \quad & L(T) - (ml - i + 1)c(v_{il-1}, v_{il}) \\ & -(ml - i)c(v_{il}, v_{il+1}) \\ & -(mh - i - 1)c(v_{ih-1}, v_{ih}) \\ & -(mh - i)c(v_{ih}, v_{ih+1}) \\ & +(ml - i + 1)c(v_{il-1}, v_{ih}) \\ & +(ml - i)c(v_{ih}, v_{il+1}) \\ & +(mh - i + 1)c(v_{ih-1}, v_{il}) \\ & +(mh - i)c(v_{il}, v_{ih+1}). \end{aligned} \tag{9}$$

□

Hence, we can calculate $L(T_i)$ by the formulation (10) in $O(1)$ time. Therefore, the complexity of exploring $N_6(T)$ is $O(ml \times mh)$.

The insert-intra-routes neighborhood considers each vertex v_i in R_l and insert it into each position in R_h. The neighborhood of R_l and R_h is defined as a set $N_7(T) = \{T_i = (R_1, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih-1}, v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{ih-1}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-$

$1, ih = 2, 3, ..., mh - 1\}$. The size of the neighborhood is $O(ml \times mh)$.

Property 7. The complexity of exploring $N_7(T)$ is $O(ml \times mh)$.

Proof. For a tour $T \in N_7(R)$, we have

$$\begin{aligned} L(T_i) &= L(T) - (ml - i)c(v_{il}, v_{il+1}) \\ &\quad -(mh - i + 1)c(v_{ih-1}, v_{ih}) \\ &\quad -(mh - i)c(v_{ih}, v_{ih+1}) - \sum_{k=1h}^{ih-1} c(v_k, v_{k+1}) \\ &\quad +(ml - i)c(v_{il}, v_{ih}) \\ &\quad +(mh - i - 1)c(v_{ih}, v_{il+1}) \\ &\quad +(mh - i + 1)c(v_{ih-1}, v_{ih+1}) \\ &\quad + \sum_{k=1l}^{il-1} c(v_k, v_{k+1}). \end{aligned} \tag{10}$$

□

Hence, we can calculate $L(T_i)$ by the formulation (11) in $O(max(mh, ml))$ time. Therefore, the complexity of exploring $N_7(T)$ is $O(max(mh, ml) \times ml \times mh)$.

5 Algorithmic design

The proposed algorithm includes two phases as follows: In the construction phase, the algorithm [9] allows a controlled amount of randomness to overcome the behavior of a purely greedy heuristic. It is used to build an initial solution for our algorithm. In the improvement phase, the penalty VNS [14] is combined with shaking operators to escape from local optima. The proposed algorithm is repeated a number of times, and the best solution found is reported. An outline of the algorithm is shown in Algorithm 1. In Step 1, the algorithm starts with an initial solution. In Step 2, it is explored switches between different neighborhoods. To explore new promising solution spaces, a diversification step is added in Step 3. In the remaining of this section, more details about the three steps of our algorithm are given.

5.1 Feasible solution space

Penalty method is a technique to solve optimization problem with constraints. It adds a penalty value to the original objective function. The advantage of penalty technique is simple to implement. It is used to solve successfully many problem [21]. All infeasible solutions are penalized by a value. With a tour T, let $V(T)$ be the violation. The violation value $V(T)$ is computed as follows:

$$V(T) = \sum_{l=1}^{k} \max\{|R_l| - L, 0\} + \sum_{l=1}^{k} \max\{K - |R_l|, 0\}.$$

Solutions are then evaluated according to the weighted fitness function $L^{'}(T) = L(T) + \rho * V(T)$, where ρ is the penalty parameter

Algorithm 1 The Proposed Algorithm

Input: $v_1, V, N_i(T)(i = 1, ..., 7), level$ are a starting vertex, the set of vertices in K_n, the set of neighborhoods and the number of swap, respectively.
Output: The best solution T^*.
Step 1 (the construction phase):
{Initially, T is an empty tour}
repeat
$T = \phi$;
for $(l = 1; l < k; l++)$ **do**
$R_l = R_l \cup v_1$; {main depot is v_1}
while (all vertices are not visited) **do**
{Pick a random route that still satisfies the constraint if the new insertion is occurred}
$R = \{R_l | R_l \in T$ and $L - 1 \leq |R_l|$ && $|R_l| \leq K - 1\}$;
if $\exists! R_l$ **then**
R = Choose a random route ($R_l \in T$) with minimum cost; {accept an invalid route}
Create a *RCL* of v_e;{v_e is the last vertex of R_l} Select a randomly vertex $v = \{v_i | v_i \in RCL$ and v_i is not visited $\}$ to add to R_l;
for $(l = 1; l \leq k; l++)$ **do**
$T = T \cup R_l$;{update the tour T}
$LT = LT \cup T$;{LT is stored a list of solutions}
until iter
T = The best feasible solution if any. Otherwise the best infeasible one in *LT*;
while stop criteria not met **do**
Step 2 (the improvement phase):
for $i : 1 \rightarrow 7$ **do**
$T^{'} \leftarrow argmin_{T^{''} \in N_i(T)} L(T^{''})$
if $((L(T^{'}) < L(T))$ or $(L(T^{'}) < L(T^*)))$ **then**
$T \leftarrow T^{'}$
if $(L(T^{'}) < L(T^*))$ and ($T^{'}$ is feasible) **then**
$T^* \leftarrow T^{'}$
else
$i++$
Step 3 (Diversification):
type = rand(2);{Select randomly a number from 1 to 2}.
if type==1 **then**
R_l = Select randomly a route $\in T$;
R_l = shaking-single-route$(R_l, level)$;
else
(R_l, R_h) = Select randomly two routes of T;
(R_l, R_h) = shaking-multi-routes$(R_l, R_h, level)$;
Update T;
return T^*;

Algorithm 2 shaking-single-route$(R_l, level)$

Input: $R_l, level$ are the $l - th$ route, and the number of swap, respectively.
Output: a new solution R_l.
while $(level > 0)$ **do**
select i, j positions from R_l at random
if $(i \neq j)$ **then**
Insert $R_l[i]$ between $R_l[j]$ and $R_l[j + 1]$;
$level \longleftarrow level - 1$;
return R_l;

Algorithm 3 shaking-multi-routes($R_l, R_h, level$)

Input: $R_l, R_h, level$ are the $l-th$, $h-th$ route, and the number of swap, respectively.
Output: a new solution R_l and R_h.
  while $(level > 0)$ **do**
    select $i-th$ and $j-th$ positions from R_l and R_h at random, respectively;
    swap $R_l[i]$ between $R_h[j]$;
    $level \longleftarrow level - 1$;
  return R_l and R_h;

5.2 The construction phase

Our construction phase is developed on the GRASP scheme in [9]. Only one iteration is performed, and one solution is found which is either feasible or not. Its steps is described in Algorithm 1. All routes are initialized with v_1 because it is a starting vertex. Each vertex of K_n is then added to the tour by using a Restricted Candidate List (*RCL*). The *RCL* of each vertex includes a number of vertices that are the closest to it. At an iteration, we find a route R_l that does not violate the constraint if a new insertion occurs. Otherwise, if we cannot find any route, then we accept the infeasibility. That means a route R_l with minimum cost in the tour is picked. Let v_e be the current last vertex of the route R_l. An unvisited vertex v is then picked randomly from the *RCL* of v_e to add to R_l. A solution is generated when all vertices of K_n are routed. The above steps are executed *iter* times to create *iter* solutions. They are stored in a *LT* list. The procedure then returns the feasible solution with minimum cost in the list if any. If it cannot produce any feasible solution, the solution with minimum cost is penalized by adding a value to the objective function.

5.3 The improvement phase

For a given current solution T, the neighborhood explores the neighboring solution space set $N(T)$ of T iteratively and tries to replace T by the best solution $T^{'} \in N(T)$. The main operation in exploring the neighborhood is the calculation of a neighboring solution's cost. In straightforward implementation, this operation requires $Tsol = O(n)$. However, by using the known cost of the current solution, we show that this operation can be done in constant time for considered neighborhoods. Thus, we speed up the running time of exploring these neighborhoods.
In a preliminary study, we realize that the efficiency of VNS algorithm relatively depends on the order in which the neighborhoods are used. Therefore, the neighborhoods are explored in a specific order based on the size of their structure, namely, from the small to large, such as the swap-adjacent, remove-insert, swap, 2-opt, or, swap-intra-route, and insert-intra-route. The time complexity of exploring the neighborhoods is reduced by choosing a random vertex, and then we are only interested in neighborhoods generated from this vertex's moves. The strategy is called "restricted". As a result, the size of the neighborhood is reduced by a factor $O(n)$. Another is "without restricted" strategy when the entire neighborhood is explored without first fixing a random vertex. The reduction of neighborhood size is also used in [19]. The aim of using two strategies is to introduce several options to run the proposed algorithm effectively.

5.4 Diversification

Shaking procedure allows to guide the search towards an unexplored part of the solution space. In this work, two types of shaking are used to give a new solution: shaking in a single route (shaking-single-route) and shaking in two (shaking-multi-routes). In shaking procedure in a single route, it selects the l-th route R_l of T and then swaps randomly several vertices for each other. In the rest, it picks two routes R_l and R_h in a random manner, and after that, exchanges randomly some several vertices in them. We finally return to Step 2 with the new solution. The shaking procedure is described in Algorithm 2 and 3.

The last aspect to discuss is the stop criterium of our algorithm. A balance must be made between computation time and efficiency. Here, the algorithm stops if no improvement is found after the number of loop (*NL*).

5.5 The time complexity

The running time of our algorithm mainly spends on exploring in VNS. In the VNS step, insert-intra-routes neighborhood consumes time, at least as well as the others. Assume that if these neighborhoods are invoked k_1 times, then the complexity of neighborhoods' exploration is $O(k_1 \times max(mh, ml) \times ml \times mh) \sim O(k_1 \times n^3)$ (in the worst case the size of mh or ml is n). It is also the theoretical complexity of our algorithm.

6 Computational results

The experiments are conducted on a personal computer, which is equipped with an Intel Pentium core i7 duo 2.10 Ghz CPU and 4 GB bytes RAM memory.

6.1 Datasets

The numerical analysis is performed on a set of benchmark problems for the mTRP and Bounded-mTSP [8, 17, 18]. As testing our algorithm on all instances would have been computationally too expensive, we implement our numerical analysis of some selected instances. In [17], R. Necula et al. propose the Bounded-mTSP instances based on the TSPLIB benchmark. Specifically, they transform TSPLIB four instances (eil51, berlin52, eil76, and rat99) by setting the number of salesmen to be, by turn, 2, 3, 5, and 7. With the aim of obtaining balanced solutions, they choose to set the bounds (L and K) on the number of vertices in each

route, by running the k-means clustering algorithm. Besides that, we add more real instances by randomly choosing some instances from TSPLIB. We divide the instances into two groups: 1) in group one (G1), the vertices are concentrated; 2) in the other (G2) the vertices are scattered.

6.2 Metrics

To evaluate our algorithm's solution quality, we need to compare it with the other metaheuristics. The main problem is that there exists no other metaheuristic reported in the literature for this problem. That means we found no previous attempts to solve the problem, neither exact nor heuristic (or metaheuristic), to compare. We try to solve exactly several small instances using some state-of-art solvers and use those results to evaluate the performance of the proposed algorithm. However, the approach only solve the problem with small instances, while metaheuristics is a suitable approach for the problem with large sizes. Therefore, we adapt the existing algorithms in [17] to compare with the proposed algorithm for the Bounded-mTRP. We define the improvement of our algorithm with respect to *Best.Sol* (*Best.Sol* is the best solution found by our algorithm) in comparison with the initial solution (*Init.Sol*), upper bound (*UB*) obtained by the GRASP and the adapted algorithm in [17], respectively. $Improv[\%] = \frac{Best.Sol - Init.Sol}{Init.Sol} \times 100\%$, and $Gap[\%] = \frac{Best.Sol - UB}{UB} \times 100\%$. In addition, we choose several state-of-the-art metaheuristic algorithms for the Bounded-mTSP [17] (Bounded Multiple Traveling Salesman Problem) and mTRP (Multiple Traveling Repairmen Problem) in [8, 18] as a baseline in our research.

6.3 Results and discussions

Through preliminary experiments, we observe that the values $iter = 10, \rho = 10, \alpha = 5$, *level*=5, *PF*=100, and *NL* = 100 resulted in a good trade-off between solution quality and run time. In this paper, the neighborhoods' order is as follows: swap adjacent, remove-insert, swap, 2-opt, or-opt, swap-intra-routes, and insert-intro-routes. These settings have thus been used in the following experiments.

In the tables, *Init.Sol*, *Best.Sol*, *Aver.Sol*, and T correspond to the initial, best, and average solution, and the average time in seconds of ten executions obtained by our algorithm, respectively. The column ACS in Tables 1 and 2 describe the best results obtained from the adapted algorithms in [17]. Figure 1 and 2 shows the evolution of the average improvement in two strategies. The values in Figures are extracted from Table 4. In Table 5, kM-ACS, g-ACS, s-ACS, gb-ACS, and sb-ACS [17] are developed on Ant Colony System (ACS) with different strategies. The proposed algorithm is tested by selecting a fixed random vertex that is labeled "restricted". Runs in which the search explores all possible moves are labeled "without restricted".

6.3.1 Experimental results for the Bounded-mTRP

The experimental results in Table 3 are the average values calculated from Table 1 and 2. In Table 3, for all instances, it can be observed that our algorithm is capable of improving the solutions in comparison with *Init.Sol*. The average improvement of our algorithm with the two strategies is about 16.94% and 13.69%, respectively. Obviously, our algorithm can obtain a significant improvement for almost instances and required small-scaled running time. Both strategies seem to work well. The neighborhood implementation with fixed random vertex uses significantly less computing time, combined with a slight loss of solution quality (about 3.25%). However, the strategy proves useful for the larger instances, for which full neighborhood search is too time-consuming. Moreover, in comparison with the algorithms in [17], the proposed algorithm also outperforms for most instances.

From Tables 5 to 6 we can draw some conclusions about the working of our algorithm. Unsurprisingly, the multi-start version of our algorithm (algorithm settings 5 to 8) requires a much larger computation time than the single-start version (settings 1 to 4). However, the quality improvement obtained by this method is relatively small. The perturbations in the GRASP+VNS algorithm (Table 6) seem to help marginally, as the solutions obtained by this algorithm are usually slightly better. This may indicate that the GRASP multi-start is not able to provide enough diversification, and that the perturbation move is useful.

For two strategies, Figure 1 and 2 shows the evolution of the average deviation to the initial solutions with respect to $\overline{improv}$ and $\overline{T}$ during the iterations in some instances. The deviations in two strategies are 14.61% (10.38%), 16.25% (11.63%), 16.48% (11.79%), 16.66% (11.94%), 16.77% (12.03%), 16.94% (13.69%), and 16.94% (13.69%) for the first local optimum, obtained by one, ten, twenty, thirty, fifty, one-hundred, and two-hundred iterations, respectively. A major part of the descent obtained by from fifty to one-hundred iterations. As can be observed, additional iterations give a minor improvement with the large running time. Hence, the first way to reduce the large running time is to use no more than one-hundred iterations, and the improvement of the proposed algorithm is about 16.94% (13.69%) for two strategies, respectively. A much faster option is to run the initial construction phase then improve it by using a single iteration, which obtains an average deviation of 14.61% (10.38%) and an average time of 0.28 (0.21) seconds.

6.3.2 Experimental results for some variants

To the best of our knowledge, most algorithms are developed for a specific variant that is not applicable to other variants. Our algorithm can be applicable to the Bounded-mTSP, although it was not designed for solving them. In comparison with the state of the art algorithms for the Bounded-mTSP, and mTRP in [8, 17, 18], our algorithm's solutions are better than the other algorithms. Specifically,

Instances	*Init.Sol*	ACS	GRASP+VNS				
			Best.Sol	*Aver.Sol*	*Improv*[%]	*Gap*[%]	*Time*
eil51_2_23_27	6214.70	5163.14	**5088.79**	5088.79	18.12	-1.44	2.50
eil51_3_15_20	4009.22	3699.57	**3466.89**	3466.89	13.53	-6.29	1.16
eil51_5_7_12	3175.37	2670.60	**2639.06**	2639.06	16.89	-1.18	0.33
eil51_7_5_10	2525.52	2828.36	**2153.82**	2153.82	14.72	-23.85	0.29
eil76_2_36_39	11832.39	10249.57	**9734.12**	9734.12	17.73	-5.03	5.08
eil76_3_21_30	8194.47	6611.05	**6937.95**	6937.95	15.33	4.94	3.71
eil76_5_12_17	5043.09	5708.21	**4252.55**	4252.55	15.68	-25.50	1.83
eil76_7_7_15	5209.80	5040.89	**4266.03**	4266.03	18.12	-15.37	1.89
eil101_2_45_57	18303.24	15942.12	**14917.5**	14917.5	18.50	-6.43	10.54
eil101_3_23_50	13618.56	11090.67	11213.95	11213.95	17.66	1.11	7.79
eil101_5_16_30	9027.06	7268.68	**7268.68**	7268.68	19.48	0.00	5.47
eil101_7_12_25	6455.90	5499.07	**5272.63**	5272.63	18.33	-4.12	5.47
Aver					17.01	-6.93	3.84
KroA100_2_49_52	686248.62	553885.22	558433.17	558433.17	18.63	0.82	13.95
KroA100_3_24_48	487994.45	417822.65	**404763.8**	404763.8	17.06	-3.13	7.44
KroA100_5_16_25	312355.55	241252.38	253177.95	253177.95	18.95	4.94	3.99
KroA100_7_11_18	255989.80	266654.12	**209079.8**	209079.8	18.32	-21.59	2.23
KroB100_2_42_59	667895.70	593219.44	**549901.5**	549901.5	17.67	-7.30	11.62
KroB100_3_27_45	458771.17	432131.45	**396202.6**	396202.6	13.64	-8.31	12.44
KroB100_5_15_27	291305.15	333112.25	**244323.25**	244323.25	16.13	-26.65	4.31
KroB100_7_11_19	244886.31	272217.45	**204165.17**	204165.17	16.63	-25.00	2.64
KroC100_2_49_52	663971.01	523174.65	556508.51	556508.51	16.18	6.37	11.95
KroC100_3_25_49	567620.77	410810.89	466018.69	466018.69	17.90	13.44	9.38
KroC100_5_12_27	360747.36	309966.89	**309055.39**	309055.39	14.33	-0.29	4.38
KroC100_7_11_22	251277.09	277550.32	**207410.95**	207410.95	17.46	-25.27	3.69
KroD100_2_46_55	709352.94	592904.12	**579919.41**	579919.41	18.25	-2.19	14.01
KroD100_5_14_29	341858.76	277429.24	277429.24	277429.24	18.85	0.00	4.94
KroD100_7_9_20	261961.85	218211.38	**217485.09**	217485.09	16.98	-0.33	1.75
KroD100_3_30_37	493176.77	395740.11	**395740.11**	395740.11	19.76	0.00	5.18
berlin52_2_10_41	87767.00	70651.94	**70651.94**	70651.94	19.50	0.00	0.86
berlin52_3_10_27	60961.19	63532.70	**50604.73**	50604.73	16.99	-20.35	1.21
berlin52_5_6_17	40292.05	34479.08	**34479.08**	34479.08	14.43	0.00	0.32
berlin52_7_4_17	36237.36	29728.2	**29728.2**	29728.2	17.96	0.00	0.47
rat99_2_46_52	35642.60	33120.70	**31375.95**	31375.95	11.97	-5.27	12.37
rat99_3_27_36	36575.94	27953.98	**22781.51**	22781.51	37.71	-18.50	8.09
rat99_5_13_30	28373.58	22458.14	**16740.8**	16740.8	41.00	-25.46	3.80
rat99_7_9_22	20659.25	14071.49	**14071.49**	14071.49	31.89	0.00	2.24
pr107_2_48_57	1247880.30	1226071.31	**1222156.88**	1222156.88	2.06	-0.32	7.57
pr107_3_25_54	1163463.77	1168230.35	**990129.24**	990129.24	14.90	-15.25	1.63
pr107_5_18_30	824379.43	793875.99	**793875.99**	793875.99	3.70	0.00	1.55
pr107_7_12_19	832903.62	887050.87	**782543.35**	782543.35	6.05	-11.78	0.41
pr124_2_44_81	1843470.97	2129804.89	**1791978.36**	1791978.36	2.79	-15.86	7.51
pr124_3_22_61	1706257.60	1580239.34	**1531978.87**	1531978.87	10.21	-3.05	6.35
pr124_5_18_42	1163970.02	1059279.15	**1017313.3**	1017313.3	12.60	-3.96	3.85
pr124_7_10_31	1139285.94	1102460.45	**919782.83**	919782.83	19.27	-16.57	2.64
Aver					16.87	-7.21	5.46

Table 1: The experimental results for Bounded-mTRP without restricted neighborhood.

Instances	*Init.Sol*	ACS	GRASP+VNS				
			Best.Sol	*Aver.Sol*	*Improv*[%]	*Gap*[%]	*Time*
eil51_2_23_27	6214.70	5163.14	**5320.66**	5320.66	14.39	3.05	2.22
eil51_3_15_20	4009.22	3699.57	**3622.07**	3622.07	9.66	-2.09	1.02
eil51_5_7_12	3175.37	2670.60	**2765.06**	2765.06	12.92	3.54	0.23
eil51_7_5_10	2525.52	2828.36	**2250.86**	2250.86	10.88	-20.42	0.12
eil76_2_36_39	11832.39	10249.57	**10165.73**	10165.73	14.09	-0.82	5.13
eil76_3_21_30	8194.47	6611.05	**7259.52**	7259.52	11.41	9.81	1.52
eil76_5_12_17	5043.09	5708.21	**4422.10**	4422.10	12.31	-22.53	0.58
eil76_7_7_15	5209.80	5040.89	**4463.30**	4463.30	14.33	-11.46	0.58
eil101_2_45_57	18303.24	15942.12	**14917.50**	14917.50	18.50	-6.43	12.02
eil101_3_23_50	13618.56	11090.67	11673.10	11673.10	14.29	5.25	3.27
eil101_5_16_30	9027.06	7268.68	**7439.82**	7439.82	17.58	2.35	1.52
eil101_7_12_25	6455.90	5499.07	**5524.72**	5524.72	14.42	0.47	1.17
Aver					13.73	-3.27	2.45
KroA100_2_49_52	686248.62	553885.22	584354.04	584354.04	14.85	5.50	11.03
KroA100_3_24_48	487994.45	417822.65	420014.20	420014.20	13.93	0.52	6.87
KroA100_5_16_25	312355.55	241252.38	**264637.54**	264637.54	15.28	9.69	3.64
KroA100_7_11_18	255989.80	266654.12	**219045.30**	219045.30	14.43	-17.85	2.14
KroB100_2_42_59	667895.70	593219.44	**574151.78**	574151.78	14.04	-3.21	10.24
KroB100_3_27_45	458771.17	432131.45	**411341.84**	411341.84	10.34	-4.81	10.64
KroB100_5_15_27	291305.15	333112.25	**254446.94**	254446.94	12.65	-23.62	3.73
KroB100_7_11_19	244886.31	272217.45	**214091.24**	214091.24	12.58	-21.35	2.21
KroC100_2_49_52	663971.01	523174.65	**584134.98**	584134.98	12.02	11.65	10.35
KroC100_3_25_49	567620.77	410810.89	487197.00	487197.00	14.17	18.59	8.68
KroC100_5_12_27	360747.36	309966.89	315991.34	315991.34	12.41	1.94	2.71
KroC100_7_11_22	251277.09	277550.32	**214251.18**	214251.18	14.74	-22.81	2.73
KroD100_2_46_55	709352.94	592904.12	602080.08	602080.08	15.12	1.55	10.35
KroD100_5_14_29	341858.76	277429.24	288911.56	288911.56	15.49	4.14	3.73
KroD100_7_9_20	261961.85	218211.38	**224589.25**	224589.25	14.27	2.92	1.14
KroD100_3_30_37	493176.77	395740.11	408633.90	408633.90	17.14	3.26	4.41
berlin52_2_10_41	87767.00	70651.94	**74100.51**	74100.51	15.57	4.88	0.65
berlin52_3_10_27	60961.19	63532.70	**51355.45**	51355.45	15.76	-19.17	1.03
berlin52_5_6_17	40292.05	34479.08	**35874.04**	35874.04	10.96	4.05	0.33
berlin52_7_4_17	36237.36	29728.2	**31158.58**	31158.58	14.02	4.81	0.36
rat99_2_46_52	35642.60	33120.70	**32886.17**	32886.17	7.73	-0.71	10.47
rat99_3_27_36	36575.94	27953.98	**23857.93**	23857.93	34.77	-14.65	6.74
rat99_5_13_30	28373.58	22458.14	**17512.63**	17512.63	38.28	-22.02	3.22
rat99_7_9_22	20659.25	14071.49	**14757.88**	14757.88	28.57	4.88	1.31
pr107_2_48_57	1247880.30	1226071.31	**1222156.88**	1222156.88	2.06	-0.32	5.06
pr107_3_25_54	1163463.77	1168230.35	**1037508.36**	1037508.36	10.83	-11.19	1.28
pr107_5_18_30	824379.43	793875.99	**828969.83**	828969.83	0.56	4.42	1.12
pr107_7_12_19	832903.62	887050.87	**814688.42**	814688.42	2.19	-8.16	0.25
pr124_2_44_81	1843470.97	2129804.89	**1869437.07**	1869437.07	1.41	-12.22	5.45
pr124_3_22_61	1706257.60	1580239.34	1592631.47	1592631.47	6.66	0.78	5.06
pr124_5_18_42	1163970.02	1059279.15	**1064491.90**	1064491.90	8.55	0.49	3.12
pr124_7_10_31	1139285.94	1102460.45	**965296.64**	965296.64	15.27	-12.44	2.45
Aver					13.64	-3.45	4.46

Table 2: The experimental results for Bounded-mTRP with restricted neighborhood.

Instances	without restricted		restricted	
	Gap[%]	*Time*	*Gap*[%]	*Time*
G1	17.01	3.84	13.73	2.45
G2	16.87	5.46	13.64	4.46
aver	16.94	4.64	13.69	3.45

Table 3: The average results for two schemes.

schemes	Dataset	1 iteration		10 iterations		20 iterations		30 iterations		50 iterations		100 iterations		200 iterations	
		Improv	*T*	*Improv*	*T*	*Improv*	*T*	*Improv*	*T*	*Improv*	*T*	*Improv*	*T*	*Improv*	*T*
without restricted	TSP-G1	13.99	0.23	16.24	0.64	16.57	0.93	16.85	1.55	17.01	3.22	17.01	3.84	17.01	14.15
	TSP-G2	15.30	0.33	16.28	0.93	16.40	1.35	16.48	2.24	16.53	4.47	16.87	5.46	16.87	19.81
	Aver	14.61	0.28	16.25	0.78	16.48	1.14	16.66	1.89	16.77	3.85	16.94	7.65	16.94	16.99
restricted	TSP-G1	11.29	0.15	13.12	0.41	13.37	0.60	13.60	0.99	13.73	2.05	13.73	2.45	13.73	9.03
	TSP-G2	9.34	0.27	9.94	0.76	10.01	1.11	10.05	1.83	10.09	3.65	13.64	4.46	13.64	16.17
	Aver	10.38	0.21	11.63	0.58	11.79	0.85	11.94	1.41	12.03	2.85	13.69	3.45	13.69	12.60

Table 4: Evolution of average deviation to *Init.Sol* without restricted neighborhood.

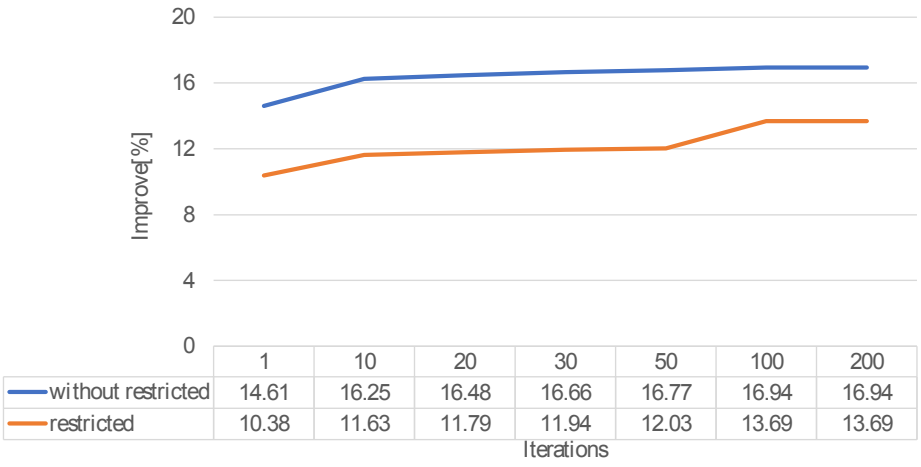

Figure 1: Evolution of average Improve [%].

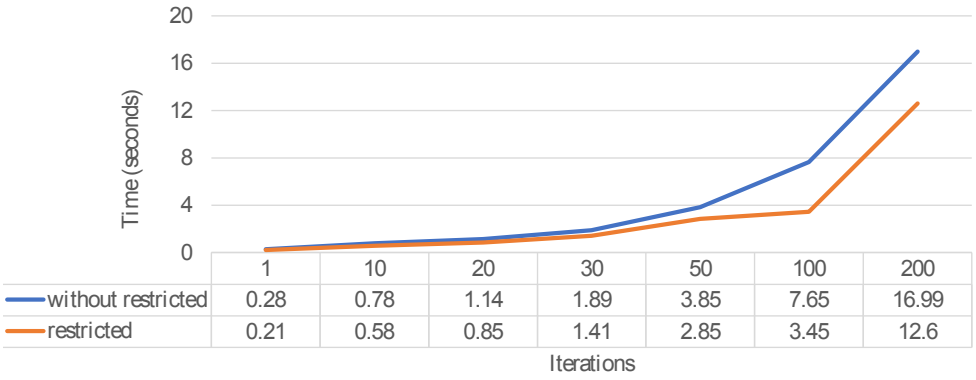

Figure 2: Evolution of average running time.

for the Bounded-mTSP in [17], our algorithm reaches better solutions for 12 out of 16 tested instances at a reasonable computational time. In addition, our algorithm can find the optimal solutions (eil51-2-23-27, eil51-3-15-20) or near-optimal solutions (berlin52-2-10-41, berlin52-3-10-27, berlin52-5-6-17) for the problems with 50 vertices

Instances	*OPT* Or [*LB, UB*]	kM-ACS	g-ACS	s-ACS	gb-ACS	sb-ACS	GRASP+VNS		
		Best.Sol	*Best.Sol*	*Best.Sol*	*Best.Sol*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	*Time*
eil51_2_23_27	442.32	454.3±0.84	452.66±1.77	454.96±2.04	452.22±1.48	453.81±1.63	**442.32**	442.32	2.36
eil51_3_15_20	464.11	500.00±0.24	485.73±3.44	489.64±3.59	479.51±3.37	483.39±3.75	**464.11**	464.11	1.26
eil51_5_7_12	[519.10, 529.70]	563.58±0.52	582.36±3.58	590.63±4.64	585.76±4.34	598.61±5.16	597.17	597.17	0.35
eil51_7_5_10	[584.02, 605.21]	634.47±0.04	674.78±4.32	680.38±3.84	688.26±3.57	699.47±4.34	731.85	731.85	0.33
berlin52_2_10_41	7753.89	8836.80±27.15	8043.92±46.91	8036.08±43.46	8057.38±43.06	8122.44±46.44	**6977.56**	6977.56	0.93
berlin52_3_10_27	8106.85	9009.18±11.83	8653.86±47.04	8806.95±64.40	8795.52±48.13	8839.37±42.63	**7942.45**	7942.45	1.32
berlin52_5_6_17	[8894.50, 9126.33]	10335.03±1.88	10164.58±90.66	10343.52±93.34	10660.46±92.58	10866.66±106.15	**8840.50**	8840.50	0.34
berlin52_7_4_17	[9415.99, 9870.02]	11966.20±2.26	11993.31±137.66	12125.55±121.75	12451.16±104.28	12712.41±97.88	**10191.53**	10191.53	0.51
eil76_2_36_39	558.59	594.21±0.93	580.77±2.91	583.41±3.04	579.68±2.39	578.96±2.81	596.09	596.09	5.52
eil76_3_21_30	579.30	642.89±0.98	622.91±3.41	630.67±5.09	613.76±3.26	619.19±4.12	**592.65**	592.65	4.04
eil76_5_12_17	[623.88, 680.67]	740.35±0.23	747.49±4.75	760.05±5.02	734.61±3.66	744.94±4.39	**685.98**	685.98	2.00
eil76_7_7_15	[675.38, 759.90]	820.35±0.10	873.65±6.61	883.63±5.44	894.70±4.68	911.06±5.00	**693.35**	693.35	2.05
rat99_2_46_52	[1296.35, 1350.73]	1485.56±2.98	1398.01±8.72	1391.89±7.35	1382.05±4.45	1389.08±5.45	1491.65	1491.65	14.55
rat99_3_27_36	[1357.30, 1519.49]	1672.11±3.33	1691.56±11.64	1707.20±12.07	1661.04±8.89	1651.68±11.57	**1449.34**	1449.34	9.30
rat99_5_13_30	[1523.95, 1855.83]	1996.04±1.23	2260.74±14.03	2297.05±16.83	2286.73±16.12	2337.94±10.45	**1810.60**	1810.60	4.38
rat99_7_9_22	[1712.1467, 2291.8207]	2361.55±0.74	2859.98±22.36	2878.97±21.11	3004.37±26.22	2984.42±17.38	**2155.53**	2155.53	2.52

Table 5: Comparisons with the state of the art metaheuristics for Bounded-mTSP without restricted neighborhood.

Instances	IOE		SNG		Our Algorithm	
	Best.Sol	*T*	*Best.Sol*	*T*	*Best.Sol*	*T*
P-n40-k5	-	-	1537.79*	0.25	1580.21	0.32
P-n45-k5	-	-	1912.31*	0.39	**1912.31**	0.37
E-n51-k5	3320	2.25	2209.64*	0.7	2247.83	0.48
P-n50-k7	-	-	1547.89*	0.70	1590.41	0.68
P-n51-k8	-	-	1448.92*	0.67	**1448.92**	0.61
E-n76-k10	4094	1.48	2310.09*	4.2	2419.89	0.91
E-n76-k14	3762	0.5	2005.4*	3.4	**2005.4**	0.83
E-n101-k8	6383	89.4	-	-	4051.47	2.94
E-n101-k14	5048	5.43	-	-	3288.53	2.92

* is the optimal value

Table 6: Comparisons with the state of the art metaheuristics for mTRP.

in several seconds in Table 5. For the mTRP in [8, 18] in Table 6, the quality of our solutions is much better than I. O. Ezzine et al.'s algorithm (IOE) in [8] and every comparable with S. Nucamendi-Guillen et al.'s algorithm (SNG) in [18]. Moreover, our algorithm can find the optimal solutions for the problems for the mTRP instance with up to 76 vertices in several seconds.

6.4 Discussions

Metaheuristic approach is a suitable approach to solve the large sized-problem. The VNS [14] is a popular schemes used widely to solve NP-hard problems. They are very effective for some variants of the mTRP problem [15, 19, 20]. The proposed algorithm is encouraged by the efficiency of the algorithms in [15, 19, 20]. However, the difference between the proposed VNS and their VNS is that our algorithm builds up penalty value during a search. Our main contribution is to adapt the VNS scheme, that extends the well known VNS by including constraint penalization, to solve the Bounded-mTRP effectively.

A good metaheuristic must balance between exploration and exploitation. Exploration is to create diverse solutions on a global space, while exploitation is to focus on the search good current local regions. In the proposed algorithm, the VNS implements exploitation while shaking maintains exploration. In terms of experiments, the proposed algorithm obtains better solutions than the adapted algorithms in [17] in many cases. We also implement two strategies that provide several choices: 1) The first choice is to run the proposed algorithm with one iteration in "restricted" strategy that obtains 10.38% solution quality on average. The running time is very fast; 2) The second is to run the proposed algorithm with one iteration in "without restricted" strategy that obtains an average improvement of 14.61%. The second option trades off solution quality and running time; 3) The last is to run the proposed algorithm no more than 100 iterations. The average improvement is 16.94%. The option is the best in terms of solution quality.

Moreover, the proposed algorithm obtains comparable or better solutions than the algorithms for the mTRP and Bounded-mTSP in [8, 17, 18]. It shows that our algorithm is still effective for various problems.

7 Conclusions

In this paper, we propose the first metaheuristic algorithm, which is mainly based on the VNS and GRASP's principles to solve the problem. Extensive numerical experiments on benchmark instances show that, on average, our algorithm leads to significant improvement. For small instacnes, our algorithm obtains the optimal solutions for the problem with 76 vertices at a reasonable a mount of time. For larger instances, the proposed algorithm reaches better solutions than the state-of-the-art algorithms in many cases.

References

[1] A. Archer, A. Levin, and D. Williamson, "A Faster, Better Approximation Algorithm For The Minimum Latency Problem", J. SIAM, Vol. 37, No. 1, 2007, pp. 1472-1498. https://doi.org/10.1137/07068151x.

[2] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, "The Complexity Of The Travelling Repairmen Problem", J. Informatique Theorique Et Applications, Vol. 20, pp.79–87. https://doi.org/10.1051/ita/1986200100791

[3] M. Avci, M.G. Avci, 2017, "A GRASP with iterated local search for the Traveling Repairman Problem with Profits", J. Comput. Ind. Eng. 113, PP. 323–332. https://doi.org/10.1016/j.cie.2017.09.032.

[4] Ha-Bang Ban, Duc-Nghia Nguyen, and Kien-Nguyen, "An Effective Metaheuristic for Multiple Traveling Repairman Problem with Distance Constraints", J. CAI, Vol. 38, 2019, pp. 1001-1034. https://doi.org/10.31577/cai_2019_4_883.

[5] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, "The Minimum Latency Problem", Proc. STOC, 1994, pp.163-171. https://doi.org/10.1145/195058.195125

[6] M.E. Bruni, P. Beraldi, S. Khodaparasti, "A Fast Heuristic for Routing in Post-Disaster Humanitarian Relief Logistics", J. Computers and Operations Research, Vol. 30, pp. 304-313, 2018. https://doi.org/10.1016/j.trpro.2018.09.033.

[7] N. Christofides, A. Mingozzi, p. Toth, "Exact Algorithms for the Vehicle Routing Problem based on Spanning Tree and Shortest Path Relaxations". j. Math Program, Vol. 20, 1981, pp. 255–282. https://doi.org/10.1007/bf01589353.

[8] I. O. Ezzine, and Sonda Elloumi, "Polynomial Formulation and Heuristic Based Approach for the k-Travelling Repairmen Problem", Int. J. Mathematics In Operational Research, Vol. 4, No. 5, 2012, pp. 503-514. https://doi.org/10.1504/ijmor.2012.048928.

[9] T.A. Feo, and M.G.C. Resende, "Greedy Randomized Adaptive Search Procedures", J. Global Opt., 1995, pp. 109–133.

[10] F. Jittat, C. Harrelson, and S. Rao, "The k-Traveling Repairmen Problem", Proc. ACM-SIAM, 2003, pp.655-664.

[11] D. S. Johnson, and L. A. Mcgeoch, "The Traveling Salesman Problem: A Case Study In Local Optimization In Local Search In Combinatorial Optimization", E. Aarts and J. K. Lenstra, Eds., pp. 215-310. https://doi.org/10.2307/j.ctv346t9c.13

[12] R. Jothi, and B. Raghavachari, "Minimum Latency Tours and The k-Traveling Repairmen Problem", Proc. LATIN, 2004, pp. 423–433. https://doi.org/10.1007/978-3-540-24698-5_46.

[13] O. Martin, S. W. Otto, and E. W. Felten, "Large-Step Markov Chains For The Traveling Salesman Problem", J. Complex Systems, Vol. 5, No. 3, 1991, pp. 299-326.

[14] N. Mladenovic, and P. Hansen, "Variable Neighborhood Search", J. Operations Research, Vol.24, No. 11 24, 1997, pp.1097-1100. https://doi.org/10.1016/s0305-0548(97)00031-2.

[15] N. Mladenovic, D. Urosevi, and S. Hanafi, "Variable Neighborhood Search for the Travelling Deliveryman Problem", J. 4OR, 2012; 11: 1-17. https://doi.org/10.1007/s10288-012-0212-1.

[16] Z. Luo, H. Qin, and A. Lim, "Branch-and-Price-and-Cut for the Multiple Traveling Repairman Problem with Distance Constraints", J. Operations Research, Vol. 234, No. 1, 2013, pp. 49-60. https://doi.org/10.1016/j.ejor.2013.09.014.

[17] R. Necula, M. Breaban, M. Raschip, "Performance Evaluation of Ant Colony Systems for the Single-Depot Multiple Traveling Salesman Problem", Proc. HAIS, vol. 9121, pp. 257-268, 2015. https://doi.org/10.1007/978-3-319-19644-2_22.

[18] S. Nucamendi-Guillén, I. Martínez-Salazar, F. Angel-Bello, and J. M. Moreno-Vega, "A Mixed Integer Formulation and An Efficient Metaheuristic Procedure for the k-Travelling Repairmen Problem", J. JORS, Vol. 67, No. 8, 2016, pp. 1121-1134. https://doi.org/10.1057/jors.2015.113.

[19] A. Salehipour, K. Sorensen, P. Goos, and O.Braysy, "Efficient GRASP+VND and GRASP+VNS metaheuristics for the Traveling Repairman Problem", J. Operations Research, 2011, Vol. 9, No. 2, 189-209. https://doi.org/10.1007/s10288-011-0153-0.

[20] M. Silva, A. Subramanian, T. Vidal, and L. Ochi, "A simple and effective metaheuristic for

the Minimum Latency Problem", J. Operations Research, Vol. 221, No. 3, 2012, pp.513-520. DOI: 10.1016/j.ejor.2012.03.044

[21] A. Piwonska, F. Seredynski, “A Genetic Algorithm with a Penalty Function in the Selective Travelling Salesman Problem on a Road Network. Proc. IPDPS, 2011. pp. 381-387. https://doi.org/10.1109/ipdps.2011.177.