Journal of Computer Science and Cybernetics, V.33, N.3 (2017), 272–288
DOI 10.15625/1813-9663/33/3/10511

A GRASP+VND ALGORITHM FOR THE MULTIPLE TRAVELING REPAIRMEN PROBLEM WITH DISTANCE CONSTRAINTS

HA BANG BAN

School of Information and Communication Technology, Hanoi University of Science and Technology; BangBH@soict.hust.edu.vn

Abstract. Multiple Traveling Repairmen Problem (MTRP) is a class of NP-hard combinatorial optimization problems. In this paper, an other variant of MTRP, also known as Multiple Traveling Repairmen Problem with Distance Constraint (MTRPD), is introduced. In MTRPD problem, a fleet of vehicles serves a set of customers. Each vehicle which starts from the depot is not allowed to travel any distance longer than a limit and each customer must be visited exactly once. The goal is to find the order of customer visits of all vehicles that minimizes the sum of all vertices' waiting time. To the best of our knowledge, the problem has not been studied much previously, even though it is a natural and practical extension of the Traveling Repairman Problem or Multiple Traveling Repairmen Problem case. In our work, we propose a metaheuristic algorithm which is mainly based on the principles of Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND) to solve the problem. The GRASP is used to build an initial solution which is good enough in a construction phase. In a cooperative way, the VND is employed to generate diverse neighborhoods in an improvement phase, therefore, it can help the search escape from local optimal. Extensive numerical experiments on 321 benchmark instances show that our algorithm can find the optimal solutions with up to 50 vertices in several instances. For larger instances, our algorithm obtains provably near-optimal solutions, even for large instances.

Keywords. Multiple Traveling Repairmen Problem with Distance Constraints (MTRPD), GRASP, VND, metaheuristic.

1. INTRODUCTION

The Traveling Repairman Problem (TRP) or Minimum Latency Problem (MLP) has been studied in a number of previous work [1, 2, 3, 4, 5, 6, 7, 8, 9, 14]. It has arised many practical applications, e.g., whenever repairman or servers have to accommodate a set of requests so as to minimize their total (or average) waiting time [1, 8, 14]. A generalization of the TRP is known as the Multiple Traveling Repairmen Problem (MTRP) that consists of several vehicles instead of one vehicle [17, 19]. Practical applications of MTRP can be found in Routing Pizza Deliverymen, and Scheduling Machine Problem to minimize mean flow time for jobs [14, 22]. In this work, we study an extension of the MTRP (Multiple Traveling Repairmen Problem with Distance Constraints - MTRPD) by involving a distance constraint that the route length (or maximum duration (MD)) of each vehicle cannot exceed a predetermined limit [20]. This type of constraint usually stems from regulations on working hours for workers. Informally, in the MTRPD problem, a fleet of vehicles serves a set of

© 2017 Viet Nam Academy of Science & Technology

customers. Each vehicle which starts from the depot is not allowed to travel a distance longer than any limit and each customer must be visited exactly once. The goal is to find the order of customer visits of all vehicles that minimizes the sum of waiting time of all customers.

This variant is at least as hard as the TRP and MTRP, and it is also NP-hard problem because it is a generalization of the TRP and MTRP. In this paper, we consider the problem in the metric case, and formulate the MTRPD as follows:
Given a complete graph K_n with the vertex set $V = \{1, 2, ..., n\}$, a symmetric distance matrix $C = \{c(i, j) \mid i, j = 1, 2, ..., n\}$, where $c(i, j)$ is the distance between two vertices i and j, and a predetermined limit L. Let $R = (1, 2, ..., k)$ be a set of k vehicles which begin at the main depot v_1. Suppose that the tour $T = (R_1, ..., R_l, ..., R_k)$ is a set of obtained routes from k vehicles. Let $R_l = (v_1, ..., v_h, ..., v_m)$ $(1 < m \leqslant n)$ be a route of vehicle $l(l \in k)$. Denote $P(v_1, v_h)$ is the path from v_1 to v_h on the route R_l and $l(P(v_1, v_h))$ is its length. The waiting time of a vertex v_h $(1 < h \leqslant m)$ on R_l is the length of the path from starting vertex v_1 to v_h as follows

$$l(P(v_1, v_h)) = \sum_{i=1}^{h-1} c(v_i, v_{i+1}). \tag{1}$$

The waiting time of the route R_l is defined as the sum of waiting times of all vertices in this route and its length cannot exceed a predetermined limit *MD*

$$W(R_l) = \sum_{h=2}^{m} l(P(v_1, v_h)), \tag{2}$$

$$L(R_l) = \sum_{i=1}^{m-1} c(v_i, v_{i+1}) \leqslant MD. \tag{3}$$

The total waiting time of T is the sum of all the vertices' waiting times

$$W(T) = \sum_{l=1}^{k} W(R_l). \tag{4}$$

The MTRPD asks for a $k-$ routes, which starts at a given vertex v_1, visits each vertex in the graph once exactly and the total waiting time of all vertices is minimized.

For NP-hard problems, there are three common approaches to solve, namely, 1) exact algorithms, 2) approximation algorithms, 3) heuristic algorithms. Firstly, the exact algorithms guarantee finding the optimal solution and take exponential time in the worst case, but they often run much faster in practice. However, the exact algorithms only solve with up to 50 vertices [2, 5, 20, 26]. Secondly, the approximation algorithm produces a solution within some factor α of the optimal solution. In this approach, the best approximation factor is still far from the optimal solution [1, 8, 9, 15, 17]. Thirdly, heuristic algorithms perform well in practice and validate their empirical performance on an experimental benchmark of interesting instances. Our metaheuristic algorithm depends on this approach.

To the best of our knowledge, the problem has not been well studied previously, even though it is a natural and practical extension of the Traveling Repairman Problem or Multiple Traveling Repairmen Problem case [7, 12, 22, 23, 25]. Recently, Z. Lou et al. [20] has introduced the MTRPD problem and proposed an exact algorithm for the problem. Their algorithm can solve exactly with up to 50 vertices. However, the small size of the problem is not suitable for the practical situations. In this work, we presents the first metaheuristic approach for this problem. In our work, we propose a metaheuristic algorithm which is mainly based on the principles of Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND) to solve the problem. The GRASP is used to build an initial solution which is good enough in a construction phase. In a cooperative way, the VND is employed to generate diverse neighborhoods in an improvement phase, therefore, it can help the search to escape from local optimal. Extensive numerical experiments on 321 benchmark instances show that our algorithm can find the optimal solutions with up to 50 vertices within reasonable running time. For larger instances, our algorithm obtains provably near-optimal solutions, even for large instances.

The rest of this paper is organized as follows. Section 2 presents the proposed algorithm. In section 3, and 4 we discuss computational evaluations, and discussions, respectively. The conclusions of this paper are summarized in Section 7.

2. OUR ALGORITHM

Our algorithm consists of two phases such as a Greedy Randomized Construction phase and a Variable Neighborhood Descent improvement phase. Firstly, GRASP (Greedy Randomized Adaptive Search Procedure) [13] is used to allow a controlled amount of randomness to overcome the behaviour of a purely greedy heuristic. Secondly, VND [21] is based on the principle of systematically exploring several different neighborhoods. It provides diverse neighborhoods for our algorithm, combined with shaking technique to escape from local optima. Our 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 obtained from the GRASP [13]. In Step 2, we investigate a novel neighborhoods' structure in VND and explore systematically switches between six neighborhoods. In order to extend the solution space of the problem, a diversification step is described in step 3. In the remaining of this section, more details about the three steps of our algorithm are given.

Step 1: The GRASP is used to generate an initial solution [13]. In this step, a feasible solution T is built, one vertex at a time for each route. At next step, a random route is selected from k routes in tour T, then, a restricted candidate list (RCL) is determined by ordering all non selected vertices in term of a greedy function that measures the benefit of including them in the route. After that, one element will be chosen from RCL if its addition into the route does not make the length of the new route be more than MD. Otherwise, a random vertex from V is chosen when the addition does not violated the length limit. Since all vertices are visited, the algorithm stops and the initial solution is returned. The size of RCL is a parameter of the GRASP that controls the balance between greediness and randomness. The GRASP for our algorithm is described in Algorithm 2.

If it can not find any feasible solution in this phase then we will choose the initial solution

Algorithm 1 The Proposed Algorithm

```
Input: $v_1, V, N_i(T)(i = 1, ..., 5), pos$ 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 GRASP(Generate an initial solution):
  $T = \text{GRASP}(v_1, V, k, \alpha)$;
  while stop criteria not met do
    Step 2 (VND):
    for $(i = 1; i \leqslant 6; i++)$ do
      $T^{'} = \text{argmin}_{T^{''} \in N_i(T)} L(T^{''})$
      if $((W(T^{'}) < W(T))$ or $(W(T^{'}) < W(T^*)))$ then
        $T = T^{'}$
      end if
      if $(W(T^{'}) < W(T^*)$ and $(T^{'}$ must be an feasible solution) then
        $T^* = T^{'}$
      else
        $i++$
      end if
    end for
    step 3 (Implement Diversification): {Select randomly a number from 1 to 2}
    type = rand(2);
    if (type==1) then
      $R_l$ = Select randomly the $l-th$ route of $T$;
      $R_l$ = Shaking-one-route$(R_l, pos)$;
    else
      $R_l$ and $R_h$ = Select randomly two routes of $T$;
      $R_l$ and $R_h$ = Shaking-two-routes$(R_l, R_h, pos)$ ;
    end if
    Go to step 2;
  end while
  return $T^*$;
```

based on the value of the new objective function W'

$$W' = W + PF \times \max\{LT - MD, 0\}, \tag{5}$$

where LT and PF are the longest route in the current solution (feasible or infeasible) and the penalty factor, respectively. If the obtained solution is feasible then $LT \leqslant MD$ and $W' = W$. Otherwise, an infeasible solution will be chosen based on the smallest amount of infeasibility.

Step 2: In this step, six neighborhoods investigated are divided into two types: 1-route, and 2-route. 1-route is used as a post-optimizer on single vehicle routes which consists of applying remove-insert, swap-adjacent, swap, 2-opt [18]. Meanwhile, solution improvements can often be obtained by move vertices belonging to two or more different routes in 2-route. For a given current solution T, neighborhood explores the neighboring solution space set

Algorithm 2 GRASP(v_1, V, k, α)

```
Input: $v_1, V, k, \alpha$ are a starting vertex, the set of vertices in $K_n$, the number of vehicles,
  and the size of $RCL$, respectively.
Output: the initial solution $T$. {$T$ is an initial Tour}
  $T = \phi$;
  for ($l = 1; l < k; l++$) do
    $R_l = R_l \cup v_1$; {The $l-th$ route of the tour $T$ starts at a main depot $v_1$}
  end for{$L$ is the list of visited vertices in $K_n$}
  $L = \phi$;
  while $|L| < n$ do
    $l$ = random($k$);{Choose a route randomly in $k$ routes}
    Create RCL with $\alpha$ vertices $v_i \in V$ closest to $v_t$; {$v_t$ is the last vertex in route $R_l$}
    $FND = 0$;
    while $\exists v$ is not considered in $RCL$ do
      Select a vertex $v = \{v_j | v_j \in RCL$ and $v_j \notin R_l\}$;
      if $L(R_l \cup v) \leq MD$ then
        $R_l = R_l \cup v$;
        $FND = 1$; {If we find successfully $v$ to add to $R_l$}
      end if
    end while
    if $FND == 0$ then
      Select randomly a vertex $v$ in $K_n$ such that $v \notin R_l$ and $L(R_l \cup v) \leq MD$;
      $R_l = R_l \cup v$;
    end if
    $L = L \cup v$;
  end while
  for ($l = 1; l < k; l++$) do
    $T = T \cup R_l$;{Update $l-th$ route in the tour $T$}
  end for
  return $T$;
```

Algorithm 3 Shaking-one-route(R_l, pos)

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

Algorithm 4 Shaking-two-routes(R_l, R_h, pos)

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

$N(T)$ 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 solutions' cost. In straightforward implementation in the worst case, this operation requires $Tsol = O(n)$. We describe more details about six neighborhoods as follows:

For 1-route: Assume that R_l and $|R_l|$ are a singe route and its length in T, respectively (In the worst case, $|R_l| = n$). 1-route is used only to optimizer on single route. We describe four neighborhoods' structure in turn.

- **The swap-adjacent** (see in Fig. 1 in [28]) attempts to swap each pair of adjacent vertices in R_l. The complexity of exploring the neighborhood is $O(Tsol \times |R_l|)$.
- **The remove-insert** (see in Fig. 2 in [28]) considers each vertex v_i in R_l and to places the vertex furthest away from this vertex v_i at the end of the route R_l. The complexity of exploring the neighborhood is $O(Tsol \times |R_l|)$.
- **The swap** (see in Fig. 3 in [28]) tries to swap the positions of each pair of vertices in the single route R_l. The complexity of exploring the neighborhood is $O(Tsol \times |R_l|^2)$.
- **The 2-opt** (see in Fig. 4 in [28]) removes each pair of edges from the route R_l and reconnects the vertices. The complexity of exploring the neighborhood is $O(Tsol \times |R_l|^2)$.

For 2-route: Assume that R_l, $|R_l|$ and R_h, $|R_h|$ are two different routes and their size of them in T, respectively. 2-route is used to move vertices in two different routes as follows:

- **The swap-two-routes** (see in Fig. 5 in [28]) tries to exchange the positions of each pair of vertices in R_l and R_h. The complexity of exploring the neighborhood is $O(Tsol \times |R_l| \times |R_h|)$.
- **The insert-two-routes** (see in Fig. 6 in [28]) considers each vertex v_i in R_l and insert it into each position in R_h. The complexity of exploring the neighborhood is $O(Tsol \times |R_l| \times |R_h|)$.

In preliminary study, we realize that the efficiency of VND 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 small to large, such as swap-adjacent, remove-insert, swap, 2-opt, swap-two-routes, insert-two-routes.

Step 3: Shaking procedure allows our algorithm to guide the search towards an unexplored part of the solution space. In this work, two categories are used to obtain a new solution such as shaking in a single route (Shaking-one-route) and shaking in two routes (Shaking-two-routes). In Shaking-one-route, it selects the l-th route R_l of T and then swaps randomly several vertices. In Shaking-two-routes, it picks randomly two routes R_l and R_h and then, exchanges randomly some several vertices between them. We finally return to step 2 with the new solution. The Shaking procedure is described in Algorithm 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).

The running time of our algorithm mainly spends for exploring in VND. In VND, swap-two-routes and insert-two-routes, in which their time complexity is not less than those of any neighborhoods, run in $O(Tsol \times |R_l| \times |R_h|)$ time. Assume that k_1 is a number of runs of them, therefore our algorithm requires $O(k_1 \times Tsol \times |R_l| \times |R_h|)$ time.

3. COMPUTATIONAL EVALUATIONS

In order to evaluate the efficiency of a metaheuristic algorithm, its solution can be compared to 1) the optimal solution (OPT); 2) the lower bound (LB); and 3) the initial solution of the construction phase (*init.Sol*); 4) a good upper bound of the state-of-the-art metaheuristic algorithm (UB).

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 optimal solution ($gap_1[\%]$), the lower bound of the optimal solution($gap_2[\%]$), and an initial solution ($improv[\%]$) in percent respectively as follows:

$$gap_1[\%] = \frac{best.sol - OPT}{OPT} \times 100, \% \tag{6}$$

$$gap_2[\%] = \frac{best.sol - LB}{LB} \times 100, \% \tag{7}$$

$$improv[\%] = \frac{best.sol - init.Sol}{init.Sol} \times 100.\% \tag{8}$$

For small instances, the optimal solutions for the problem from [20] allows us to evaluate the performance of our algorithm exactly. When no optimum solution is available for large instance sizes and our algorithm is the first metaheuristic for solving the problem, our best solutions can be compared to the tight lower bound of the optimal solution in [22] or initial solutions from GRASP. Certainly, the comparision between GRASP and VND is not completely fair because they depend on two different types of algorithm. Specifically, GRASP runs only once to obtain an initial solution while VND runs many times and it terminates when there is no better solution found after a number of iterations. However, the comparision is still significant to represent the efficiency of the VND phase.

3.1. Datasets

The experimental data includes two datasets. Each instance contains the coordinate of n vertices. We divide these instances into two types: Type 1 consists of the instances in which the optimal solutions of them have been known; otherwise, they depend on type 2.

Small instances and their optimal solutions in [20] are used in our experiments. We can obtain the optimal solutions for these instances by using exact algorithms in [20]. We named them as dataset 1. It includes six TSP instances from the TSPLIB such as brd14051, d15112, d18512, fnl4461, nrw1379 and pr1002. For each TSP instance, we generate ten MTRPD instances by randomly selecting subsets of n vertices, where n = 30, 40 and 50 and one vertex was arbitrarily designated as the depot. Therefore, one hundred and fifty instances are selected in the dataset 1.

The numerical analysis was performed on a set of benchmark problems for Capacitated VRP in [27]. As testing our algorithm on all instances would have been computationally too expensive, we implemented our numerical analysis on some selected instances. Firstly, in order to eliminate size effects, problems with approximately from 50 up to 200 customers are chosen. Also, in order not to bias the results by taking "easy" or "hard" instances we randomly choose instances. Specifically, this dataset includes seven instances (CMT6, CMT7, ..., CMT14) and twelve instances (tai75, ..., tai150d). Moreover, one hundred and thirty instances from 60 to 80 vertices [22] are also used in our experiments. We gather and name them as dataset 2. In this dataset, we can obtain the optimal solutions for the instances of MTRP in [22]. These optimal solutions are the lower bound of the optimal solutions of MTRPD. Most of the instances in dataset 2, the distance constraint is not available, except several instances in dataset 2. We denote by $d_{\max}$ the greatest distance for any route involving a single customer and impose on each vehicle a travel distance limit $MD = 2 \times d_{\max}$. The similar generation for travel distance limit can be found in [10].

3.2. Results

The experiments are conducted on a personal computer, which is equipped with an Intel Pentium core i7 duo 2.10 Ghz CPU and 8 GB of bytes RAM.

Two experiments are conducted on the above datasets. For the instances in dataset 1, their optimal solutions let us evaluate exactly the efficiency of our algorithm. For the instances in dataset 2, since their optimal solutions have been not known, the efficiency of the algorithms is only evaluated relatively.

Through preliminary experiments, we observed that the values $\alpha = 5$, pos=5, PF=100, and $NL = 100$ resulted in a good trade-off between solution quality and run time. In addition, in a pilot study, the performance of the algorithm relatively depends on the order in which the neighborhoods are used. In this paper, the order of the neighborhoods is as follows: swap adjacent, remove-insert, swap, 2-opt, swap-two-routes, insert-two-routes. These settings have thus been used in the following experiments.

In Tables, *best.sol*, *aver.sol* correspond to the best solution, average solution for our metaheuristic algorithm after ten runs, respectively. Tables 1 compare the results of our algorithm with the optimal solutions in [20]. Table 3, ..., 5 compare the results of our algorithm with the lower bound of the optimal solution (note that the optimal solution of the MTRP is the lower bound of the optimal solution of the MTRPD. The optimal solution

Table 2. The average experimental results for dataset type 1

Instances	n=30		n=40		n=50	
	$\overline{gap_1}$	$\overline{T}$	$\overline{gap_1}$	$\overline{T}$	$\overline{gap_1}$	$\overline{T}$
ine brd14051-x	0.00	0.45	0.26	0.45	0.63	0.90
d15112-x	0.00	0.22	0.30	0.80	0.44	0.46
fnl4461-x	0.00	0.25	0.26	0.35	0.59	0.75
nrw1379-x	0.00	0.22	0.47	0.81	0.54	0.72
pr1002-x	0.00	0.21	0.28	0.47	0.54	0.77

Table 6. The average experimental results for dataset type 2

Instances	$\overline{gap_2}$	$\overline{improv}$	$\overline{T}$
brd14051-60-x	3.65	13.62	1.83
d15112-60-x	3.60	13.68	2.02
fnl4461-60-x	2.69	13.53	1.64
nrw1379-60-x	4.21	13.89	1.88
pr1002-60-x	4.75	13.56	1.99
brd14051-70-x	3.59	13.69	3.07
d15112-70-x	2.73	13.95	2.53
fnl4461-70-x	3.82	14.28	2.84
nrw1379-70-x	4.53	14.46	2.64
pr1002-70-x	3.67	14.19	2.82
brd14051-80-x	3.32	15.47	10.47
d15112-80-x	2.86	15.40	8.99
fnl4461-80-x	4.05	15.54	9.11
nrw1379-80-x	3.06	15.45	11.04
pr1002-80-x	3.87	15.60	9.25
tai-x	-	28.09	34.60
CMT	-	26.66	67.98

of the MTRP is obtained in [22]) and the initial solution from GRASP. Table 7 shows the evolution of the average deviation to the initial solutions during the iterations on some instances. Table 8, 9 compare our results with the state-of-the-art metaheuristic algorithms in MTRP and CCVRP, such as I. O. Ezzineet al.s (IOE) [12], L. Ke et al.s (CCVRP) [16], SU. Ngueveu (MA1) [23], S. Nucamendi-Guillen et al.s (SNG) [22], G. Riberio et als (ALNS) [25]. The optimal solutions in Table 8 are highlighted in '*' symbol.

The average experimental results are illustrated in Table 2 for all instances in Type 1, which are the average values calculated from Table 1. Similarly, the results are described in Table 6, which are the average values calculated from Table 3 to 5 for all instances in Type 2. In Table 2 and 6, we denote by $\overline{gap_1}, \overline{gap_2}, \overline{improv}$ and $\overline{T}$ the average values of $gap_1, gap_2, improv$ and T for each MTRPD dataset, respectively.

In Table 2, for all instances, it shows that our algorithm is capable of finding the optimal solutions for several instances which is up to 50 vertices in dataset 1 at a reasonable amount

Table 7. Evolution of average deviation to *Init.Sol*

Dataset	1 iterations		20 iterations		30 iterations		50 iterations		100 iterations		200 iterations	
TSP-x	11.87	0.35	13.79	0.96	14.06	1.39	14.30	2.31	14.44	4.81	14.44	8.93
tai-x	25.99	2.52	27.65	7.21	27.87	10.45	28.00	17.30	28.09	34.60	28.09	72.08
CMT	25.10	2.34	26.22	16.41	26.47	21.10	26.54	32.82	26.66	67.98	26.66	135.96
Aver	20.99	1.74	22.55	8.19	22.80	10.98	22.95	17.48	23.06	35.80	23.06	72.32

of time. In addition, it is obvious that, in average, our solutions fall within 1.0% of the optimal solutions.

Table 6, for all instances, it can be observed that our algorithm is capable of improving the solutions in comparison with *Init.Sol* and *LB*. The average improvement of our algorithm with average *improv* is between 13.52% and 28.09%. Moreover, our solutions are below 5.0% of the lower bound of the optimal solutions. Obviously, our algorithm can obtain provably near-optimal solutions, even for large instances and required small scaled running time.

Table 8. Comparsion with state of the art metaheuristic algorithms for CCVRP

Instances	*n*	*k*	**MA1**		**ALNS**		**LK**		**Our Algorithm**	
			best.sol	*T*	*best.sol*	*T*	*best.sol*	*T*	*best.sol*	*T*
CMT1	50	5	2230.35	10.63	2230.35	30.29	2230.35	17.6	2230.35	1.27
CMT2	75	10	2391.63	27.78	2421.90	60.77	2391.63	22.4	2391.63	5.76
CMT3	100	8	4045.42	449.44	4073.12	235.12	4045.42	92.6	4167.32	18.49
CMT4	150	12	4987.52	97.91	4987.52	172.45	4987.52	60.9	5178.31	81.66
CMT11	120	7	7315.87	160.64	7317.98	202.07	7314.55	71.6	7347.49	27.09
CMT12	100	10	3558.92	38.20	3558.92	152.74	3558.92	53.7	3622.12	16.14

Table 9. Comparsion with state of the art metaheuristic algorithms for MTRP

Instances	*n*	*k*	IOE		SNG		Our Algorithm		
			best.sol	*T*	*best.sol*	*T*	*best.sol*	gap_1	*T*
E-n51-k5	50	5	3320.00	2.25	2209.64*	0.7	2386.87	8.02	2.49
E-n76-k10	75	10	4094.00	1.48	2310.09*	4.2	2577.34	11.57	5.46
E-n76-k14	75	14	3762.00	0.51	2005.4*	3.4	2176.25	8.52	5.26
E-n76-k15	75	15	3822.00	0.09	1962.47*	2.81	2182.32	11.20	5.16
E-n101-k8	100	8	6383.00	89.4	-	1.47	4862.12	-	11.81
E-n101-k14	100	14	4504.00	5.40	-	10.53	3314.23	-	10.91
Aver								9.83	

In table 7, the deviations are 20.99%, 22.55%, 22.80%, 22.95%, 23.06%, and 23.06% for the first local optimum, obtained by one, ten, twenty, thirty, fifty, one-hundred, and two-hundred calls VND, respectively. 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 fifty calls to VND and the improvement of our algorithm is about 23.06%. A much faster option is to run the initial construction phase then improve it by using a single call to VND, which obtains an average deviation of 20.99% and average time of 1.74 seconds, even for the instances which are up to 200 vertices.

3.3. Comparision with MTRP and CCVRP

As we mentioned in the first section, the MTRPD is a generalization of the MTRP (Multiple Traveling Repairmen Problem) and CCVRP (Cumulative Capacity Vehicle Routing Problem). It follows that any solution method for the MTRPD can be used to solve instances of the MTRP and CCVRP. Although many researchers have studied the MTRP and CCVRP [12, 16, 22, 23, 25], the literature on the MTRPD is surprisingly limited. The only prior study we can find in existing literature is an exact algorithm [20] for the MTRPD. In this work, we present the first metaheuristic approach for this problem, therefore, there is no direct comparison to the other algorithms. However, our algoirthm can be indirectly compared to the state of the art algorithms in MTRP [12, 22] in Table 7 and CCVRP [16, 23, 25] in Table 8 (note that MTRP is a particular variant of CCVRP where the vehicle capacity restrictions are removed). In these experiments, our algorithm is run without the distance and capacity constraints. By comparing the other algorithms, it becomes clear that, although not designed for it, our algorithm still performs better than IOE for the most of instances and nearly as good as the other MTRP and CCVRP metaheuristics. In particular, our solutions are very close to the optimal solutions since the solutions are within 9.83% of the optimum in MTRP and comparable to the others in CCVRP.

The algorithms are implemented on computers with different configurations. Therefore, we only evaluate the running time of them relatively. In Table 8 and 9, the running time of our algorithm is comparable to those of the others.

4. DISCUSSIONS

Three types of algorithms are used to solve MTRPD. The first type consists of exact algorithms that are time-consuming, however, they can only solve the problem with small sizes. The second type consists of classical heuristics such as greedy, local search, relaxation based etc. These heuristics produce approximate solution faster, when compared with the first type, but without guarantee of optimality. The third type consists of heuristics that are based on some metaheuristic rules. Our algorithm depends on metaheuristic.

In combinatorial optimization problem such as MTRPD, there exist many "locally optimal", but enough good as well as global optimal solution. To the best of our knowledge, algorithms often get trapped into local optimum because their explored part of the solution space is not large enough. Our metaheuristic approach takes the advantages of both GRASP and VND. Firstly, GRASP is a constructive heuristic that starts with an empty solution and repeatedly extends the current solution until a complete solution is obtained. The main purpose of the implemented construction heuristics is to feed the main algorithm with an initial solution, but it also meets performance comparison purposes. Secondly, in the improvement phase, a metaheuristicVND tries to improve the current solution further via a set of neighborhood moves. Even though the initial solution is set far from global minima, VND still helps our algorithm to extend the explored part of the solution space in order to escape local minima, and thus drives our search to global minima. Our algorithm performs better than local search because heuristic approaches (gradient method, local search, ...) are often too greedy, therefore, they often get stuck in local optimum. Our metaheuristic approach, on the other hand, is not greedy and allows them to explore more thoroughly the solution space, thus the chance to obtain better solutions is higher than heuristics. In addition, it is very simple and fast to implement. However, better solutions found in VND phase only is accepted when their distance constraint is satisfied (or their solutions are valid), therefore, the quality of solution cannot be improved after VND phase in some special cases. Our experiments show that the average improvement obtained by VND falls into the range of 13.52% and 28.09% in comparison with GRASP. That improvement presents the efficiency of the VND phase. In addition, our algorithm also is compared to the other algorithms in MTRP and CCVRP. Currently, there have been a few published studies [12, 16, 22, 23, 25] with metaheuristics developed for the MTRP and CCVRP. In the experiment, we run our algorithm without the distance and capacity constraint, therefore, it can directly compare to the state-of-the-art metaheuristics. The experimental results show that our algorithm significantly performs nearly as good as the state-of-the-art algorithms, although our algorithm is not designed for these problems.

Our algorithm in this paper and the algorithm in [7] also use the popular scheme of the combination of GRASP and VND. The differences between them come from some points as follows: 1) our problem is a generation of the problem in [7], therefore it becomes more difficult to solve. In [7], any solution found is valid, thus their algorithm always returns a proper solution while we cannot guarantee that our algorithm always obtains a valid solution because of the distance constraint. Even so, it is difficult to find a valid solution in many cases; 2) In combinatorial optimization problems, GRASP, and VND are the popular and effective techniques to solve. However, GRASP in our work is modified by using a penalty factor. Similarity, in VND, moves from there to a new one if and only if an improvement was made and new solution obtained must be valid. It can be said that our algorithm is the development from the algorithm in [7] for the more general case; 3) benchmark in our work is different from the one in [7] in terms of the presence of the distance constraint.

5. CONCLUSIONS

In this paper, we propose the first metaheuristic algorithm which is mainly based on the principles of VND and GRASP to solve the problem. Moreover, we introduce a new novel neighborhoods structure in VND for the problem. Extensive numerical experiments on benchmark instances show that our algorithm can find the optimal solutions with up to 50 vertices at a reasonable amount of time. For large instances, our algorithm led to significant improvement between 13.52% and 28.09% and also required small computational time, even 20.99% with an average time of 1.74 seconds which is only used for a single call to VND. In comparison with the state of the art metaheuristics in MTRP and CCVRP, our computation time as well as the solution quality are comparable although our algorithm is not designed for them. In the future, we intend to extend our algorithm by including more neighborhoods and careful study of the effectiveness of each neighborhood on the problem. Enhancing the efficiency of our algorithm to allow even larger problems to be solved, is another future research topic.

ACKNOWLEDGEMENT

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number DFG 102.01-2016.03.
The experimental results in this research are implemented by the computing resource from Data Science Laboratory and Modeling, Simulation, Optimization Lab, School of Information and Communication Technology, Hanoi University of Science and Technology.

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, pp. 1472–1498, 2007.

[2] H.G. Abeledo and G. Fukasawa and R. Pessoa and A. Uchoa, "The time dependent traveling salesman problem: Polyhedra and branch-cut-and price algorithm", *J. Math. Prog. Comp.*, vol. 5, no. 1, 2013, pp. 27-55.

[3] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, N. and Papakostantinou, "The complexity of the travelling repairman problem", *J. Informatique Theorique et Applications*, vol. 20, no.1, pp.79–87, 1986.

[4] H.B. Ban, and D.N. Nguyen, "Improved genetic algorithm for minimum latency problem", *Proc. SOICT*, pp. 9–15, 2010.

[5] H.B. Ban, K. Nguyen, M.C. Ngo, and D.N. Nguyen, "An efficient exact algorithm for minimum latency problem", *J. PI*, vol. 1, no.10, pp. 1-8, 2013.

[6] H.B. Ban, and D.N. Nguyen, "A meta-heuristic algorithm combining between tabu and variable neighborhood search for the minimum latency problem", *Proc. RIVF*, pp. 192–197, 2013.

[7] H.B. Ban, and D.N. Nguyen, "An effective GRASP+VND metaheuristic for the k-minimum latency problem", *Proc. RIVF*, pp. 192–197,2016.

[8] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem", *Proc. STOC*, pp.163–171, 1994.

[9] K. Chaudhuri, B. Goldfrey, S. Rao, and K. Talwar, "Path, tree and minimum latency tour", *Proc. FOCS*, pp. 36–45, 2003.

[10] L. Chung-Lun, D. Simchi-Levi, M. Desrochers, "On the distance constrained vehicle routing problem", *J.Operations Research*", vol.4, no. 4, pp. 790 – 799, 1992.

[11] A. L. Erera, J. C. Morales, and M. W. P. Savelsbergh, "The vehicle routing problem with stochastic demand and duration constraints", *J. Transportation Science*, vol. 44, no. 4, pp. 474 - 492, 2010.

[12] I. O. Ezzine, Sonda Elloumi, "Polynomial formulation and heuristic based approach for the k-travelling repairmen problem", *Int. J. Mathematics in Operational Research*, vol. 4, no. 5, pp. 503–514, 2012.

[13] T.A. Feo and M.G.C. Resende, “Greedy randomized adaptive search procedures”, *J. Global Opt.*, pp. 109-133, 1995.

[14] M. Fischetti, G. Laporte, and S. Martello, “The delivery man problem and cumulative matroids, *J. Operations Research*, vol. 41, pp. 1055 - 1064, 1993.

[15] M. Goemans and J. Kleinberg, “An improved approximation ratio for the minimum latency problem”, *Proc. SIAM SODA*, pp. 152–158,1996.

[16] L. Ke and Z. Feng, A two-phase metaheuristic for the cumulative capacitated vehicle routing problem, *J. Computers and Operations Research*, vol. 40, no. 2, pp. 633–638, 2013.

[17] F. Jittat and C. Harrelson and S. Rao, “The k-traveling repairmen problem”, *Proc. ACM-SIAM*, pp.655–664,2003.

[18] 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., Wiley, New York, pp. 215–310, 1997.

[19] R. Jothi, B. Raghavachari, “Minimum latency tours and the k-traveling repairmen problem”, *Proc. LATIN*, pp. 423-433, 2004.

[20] Z. Luo, H. Qin and A. Lim, “Branch-and-price-and-cut for the multiple traveling repairmen problem with distance constraints”, *J. Operations Research*, vol. 234, no. 1, pp. 49–60, 2013.

[21] N. Mladenovic, P. Hansen, “Variable neighborhood search”, *J. Operations Research*, vol.24, no. 11 24, pp.1097–1100, 1997.

[22] S. Nucamendi-Guilln, I. Martnez-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, pp. 1121–1134, 2016.

[23] SU. Ngueveu, C. Prins and R. Wolfler Calvo, “An effective memetic algorithm for the cumulative capacitated vehicle routing problem”, *J. Computers and Operations Research*, vol. 37, no. 11, pp. 1877–1885, 2010.

[24] D. Simchi-Levi and O. Berman, “Minimizing the total flow time of n jobs on a network”, *J. IIE Transactions,* vol. 23, no. 3, pp. 236–244,1991.

[25] G. Ribeiro, G. Laporte, “An adaptive large variable neighborhood search heuristic for cumulative capacitated vehicle routing problem”, *J. Computers and Operations Research,* vol. 39, no. 3, pp. 28–35.

[26] B.Y. Wu, Z.-N. Huang and F.-J. Zhan, “Exact algorithms for the minimum latency problem”, *Inform. Proc. Letters*, vol. 92, no. 6, pp. 303–309, 2004.

[27] http://neo.lcc.uma.es/vrp/vrp-flavors/capacitated-vrp/

[28] https://sites.google.com/site/eightneighborhoods/

Received on July 12, 2017
Revised on November 17, 2017

APPENDIX

Table 1. The experimental results for dataset type 1

Instances		*n*=30, *k*=6				*n*=40, *k*=8				*n*=50, *k*=10			
		OPT	*best.sol*	*aver.sol*	*T*	*OPT*	*best.sol*	*aver.sol*	*T*	*OPT*	*best.sol*	*aver.sol*	*T*
pr1002	0	168188.80	168188.80	272908.40	0.19	233387.20	234559.70	234928.65	0.38	272908.40	274612.40	274874.47	0.72
	1	195805.80	195805.80	249275.40	0.18	218781.40	218781.40	219715.60	0.38	249275.40	251158.80	251564.40	0.73
	2	182635.00	182635.00	277959.10	0.20	211241.70	212136.50	212365.37	0.40	277959.10	278887.60	279113.61	0.69
	3	139784.70	139784.70	298846.10	0.18	196120.40	197071.80	197230.69	0.42	298846.10	299799.40	299982.79	0.72
	4	164916.70	164916.70	262518.40	0.17	227450.10	227450.10	227648.64	0.50	262518.40	262518.40	264323.66	0.68
	5	163642.30	163642.30	273318.80	0.18	194802.20	195630.00	195799.75	0.43	273318.80	277004.80	277255.46	0.68
	6	160585.90	160585.90	280317.30	0.19	229730.10	229730.10	231345.39	0.48	280317.30	280317.30	282363.62	0.72
	7	166887.00	166887.00	246341.00	0.20	236896.10	237187.80	237204.56	0.39	246341.00	248305.70	248446.62	0.73
	8	161025.80	161025.80	256971.40	0.18	230126.60	230679.10	230904.24	0.47	256971.40	258526.80	259101.37	0.74
	9	144167.00	144167.00	267596.70	0.18	192179.90	193276.50	193463.13	0.48	267596.70	269132.50	269446.87	0.75
brd14051	0	97380.30	97380.30	133642.30	0.19	97630.70	97777.80	97873.15	0.43	133642.30	135037.10	135190.87	0.68
	1	96322.50	96322.50	123212.70	0.18	110671.10	110935.00	110977.16	0.42	123212.70	123592.50	123646.28	0.69
	2	64109.40	64109.40	137175.10	0.19	127629.70	127629.70	127802.82	0.46	137175.10	137795.10	137864.68	0.75
	3	89582.50	89582.50	150209.80	0.18	99527.60	99764.80	99800.43	0.37	150209.80	151470.90	151573.55	0.75
	4	87615.70	87615.70	116278.70	0.19	123881.80	124005.60	124063.66	0.49	116278.70	117078.60	117223.95	0.75
	5	75079.50	75079.50	124648.20	0.19	98329.40	98329.40	98606.42	0.49	124648.20	124648.20	125120.08	0.69
	6	94540.80	94540.80	121190.40	0.20	110676.60	111083.00	111171.57	0.36	121190.40	122472.80	122618.57	0.72
	7	81515.80	81515.80	124077.60	0.19	103775.50	104131.50	104197.46	0.42	124077.60	124714.00	124805.31	0.68
	8	74160.80	74160.80	125446.00	0.19	101387.30	102229.30	102356.58	0.41	125446.00	126072.00	126137.35	0.72
	9	90628.10	90628.10	118925.00	0.18	87945.10	88271.10	88327.50	0.48	118925.00	119942.00	120022.13	0.75
fnl4461	0	51192.20	51192.20	81562.20	0.19	63096.00	63255.90	63289.70	0.48	81562.20	82262.50	82342.65	0.68
	1	44154.70	44154.70	79804.80	0.18	66882.60	66882.60	67347.95	0.38	79804.80	79804.80	80394.96	0.71
	2	46571.20	46571.20	73309.10	0.18	70151.40	70257.60	70277.79	0.39	73309.10	74245.20	74417.87	0.69
	3	48591.40	48591.40	79335.10	0.19	58843.90	59080.90	59167.14	0.43	79335.10	79745.70	79791.55	0.67
	4	54485.90	54485.90	75052.00	0.18	61654.90	61654.90	62092.89	0.38	75052.00	75326.90	75417.03	0.70
	5	47907.30	47907.30	76738.10	0.18	56144.50	56312.70	56349.92	0.42	76738.10	77397.10	77432.55	0.75
	6	45882.10	45882.10	75268.90	0.18	61274.90	61703.40	61805.05	0.43	75268.90	75268.90	75841.85	0.70
	7	44545.30	44545.30	72956.30	0.20	65698.30	65817.60	65904.44	0.38	72956.30	73236.80	73406.63	0.72
	8	50365.30	50365.30	70244.00	0.20	64260.90	64516.80	64546.20	0.48	70244.00	71043.50	71226.29	0.72
	9	49179.60	49179.60	82157.00	0.18	58717.50	58833.50	58922.49	0.49	82157.00	82602.50	82627.28	0.73
d15112	0	225070.20	225070.20	353657.80	0.18	287734.80	288775.30	288880.38	0.37	353657.80	356010.10	356206.86	0.73
	1	213332.30	213332.30	355115.20	0.19	256987.10	256987.10	257046.88	0.39	355115.20	357795.00	358340.92	0.72
	2	208323.60	208323.60	392196.10	0.19	307407.10	308184.10	308378.92	0.46	392196.10	394937.80	395443.60	0.75
	3	222870.40	222870.40	350821.80	0.19	292602.70	292602.70	294673.72	0.42	350821.80	352514.80	352927.54	0.72
	4	216056.00	216056.00	341493.60	0.19	299259.00	300039.10	300434.70	0.35	341493.60	342899.20	343504.91	0.74
	5	235215.80	235215.80	360717.40	0.18	269559.40	269559.40	270295.81	0.35	360717.40	360717.40	364367.19	0.72
	6	207139.00	207139.00	390251.40	0.19	320989.40	323197.50	323704.55	0.45	390251.40	393471.50	394039.37	0.72
	7	280309.00	280309.00	327701.90	0.20	287270.70	289161.90	289368.39	0.41	327701.90	327701.90	330738.28	0.73
	8	244015.40	244015.40	344600.50	0.19	303263.90	304002.60	304260.97	0.38	344600.50	345565.50	345872.08	0.68
	9	238976.20	238976.20	347783.60	0.20	282412.30	283799.60	284133.95	0.40	347783.60	348895.00	349108.87	0.72
nrw1379	0	31249.40	31249.40	39206.10	0.17	35655.20	35810.30	35847.05	0.45	39206.10	39396.10	39441.78	0.74
	1	33138.50	33138.50	61449.00	0.17	33000.70	33206.80	33264.85	0.39	44233.60	44931.60	44975.83	0.68
	2	31872.00	31872.00	45914.20	0.19	39928.10	40189.40	40212.15	0.49	45914.20	46171.20	46229.34	0.72
	3	31777.10	31777.10	46208.00	0.18	36685.40	36685.40	36822.25	0.50	46208.00	46208.00	46407.92	0.68
	4	26671.20	26671.20	43557.90	0.18	36168.60	36433.50	36494.91	0.38	43557.90	43813.90	43854.03	0.75
	5	29010.30	29010.30	46718.40	0.18	38005.40	38005.40	38365.05	0.42	46718.40	47031.00	47078.92	0.71
	6	30398.10	30398.10	49421.10	0.19	31837.30	32051.90	32071.83	0.36	49421.10	49421.10	49716.24	0.73
	7	30765.50	30765.50	49960.10	0.19	39394.80	39546.10	39604.12	0.49	49960.10	50118.70	50132.81	0.75
	8	28796.40	28796.40	41560.90	0.20	36674.50	36950.70	37026.01	0.37	41560.90	41852.50	41887.55	0.74
	9	26271.20	26271.20	44404.00	0.18	36447.70	36616.60	36679.32	0.43	44404.00	44610.90	44644.15	0.68

Table 3. The experimental results for dataset type 2 ($n = 60$, $k = 12$; $n = 70$, $k = 14$; $n = 80$, $k = 16$)

Instances		*n*=60, *k*=12					*n*=70, *k*=14					*n*=80, *k*=16				
		LB	*Init.Sol*	*best.sol*	*aver.sol*	*T*	*LB*	*Init.Sol*	*best.sol*	*aver.sol*	*T*	*LB*	*Init.Sol*	*best.sol*	*aver.sol*	*T*
pr1002	0	530946.01	624231.08	541518.09	543903.28	1.59	429557.7	511564.15	442509.8	444351.38	3.32	491764.64	605318.81	511218.66	515561.48	9.24
	1	356469.79	439998.65	377473.81	383945.49	1.46	430048.06	512027.48	441809.41	443999.19	3.09	442164.21	539934.18	457387.24	459860.99	11.76
	2	344118.14	415805.02	358419.49	361372.57	1.48	377233.86	451708.22	390907.01	392183.57	3.01	505524.17	615924	519606.67	523547.93	11.01
	3	429604.2	509777.61	441824.31	444883.13	1.68	429562.01	513336.09	443380.91	445322.95	3.08	436752.96	541018.84	456149.24	460190.28	9.37
	4	435655.25	514878	445782.01	447420.02	2.32	435659.17	519537.94	449799.03	451229.49	2.81	453609.46	565005.72	474845.52	480470.87	11.92
	5	668129.73	781751.6	678573.15	680714.14	1.99	429584.16	515233.76	445727.54	447318.79	2.59	599492.4	719244.77	610148.31	613313.66	10.19
	6	406678.77	480718.57	417915.43	420052.82	2.22	344534.44	414393.16	356928.63	359297.21	3.13	619003.36	743912.55	630654.79	632964.65	10.22
	7	311254.73	392558	335041.18	341585.67	1.93	393558.46	485548.26	415419.95	420274.31	3.65	508186.51	621569.54	526575.16	529628.12	11.69
brd14051	0	213420.42	251835.6	219201.55	220216.38	1.93	191843.35	236354.03	201653.32	203644.28	2.92	336983.07	403747.82	342913.81	344342.84	9.5
	1	218315.68	257416.84	222922.72	223968.72	1.97	169340.01	203146.37	174699.78	175298.26	2.97	277861.02	333019.1	282585.16	283312.79	8.8
	2	151666.85	181914.3	157250.92	158763.22	2.19	216195.95	257111.44	221647.94	223750.07	1.99	265370.92	324193.92	274711.39	276172.69	8.28
	3	172199.83	205385.88	177816.43	178531.28	2.28	229328.9	276102.19	237381.5	239270.5	2.62	189815.69	229602.51	194774.26	195546.91	8.22
	4	133952.5	165830.06	141589.33	143382.12	2.01	302498.42	354806.5	306360.85	307412.49	2.44	206068.45	253213.3	213799.91	214959.31	9.53
	5	203145.14	237894.93	206627.1	207747.32	2.55	179470.31	213645.57	183904.48	184788.33	2.5	303621.75	364273.38	308990.83	309759.04	9.55
	6	136233.51	167976.86	143533.26	145536.4	2.33	231693.74	274493.01	235984.81	236908.43	2.51	213405.23	266743.89	223966.26	227673.79	9.61
	7	171879.58	207856.03	179225.91	180925.64	1.82	284960.31	335936.51	290522.96	291669.13	2.11	263737.93	321992.48	271200.68	273907.99	9.04
fnl4461	0	156260.54	182219.29	158305.97	158921.09	1.54	154805.67	183733.16	157669.13	158434.84	2.73	153124.51	186691.81	158419.6	159133.94	8.72
	1	103190.13	123710.24	107064.19	107778.46	1.78	104585.82	130419.59	111849.65	112856.43	2.7	174975.64	210564.66	178780.39	179544.3	8.66
	2	109739.93	130783.36	113224.83	114231.94	1.48	161892.44	191040.61	164334.48	164989.48	3.05	162755.5	198400.84	167901.28	168902.49	9.93
	3	100792.2	120893.58	104439.75	105196.91	1.82	99122.23	121771.78	104161.61	105267.83	2.63	160819.04	199007.5	167789.06	169417.9	9.7
	4	149638.18	175744.17	152778.2	153335.82	1.66	157106.13	184661.26	159413.31	160041.12	2.48	151790.69	187481.23	157787.23	159419.8	7.49
	5	158679.44	186523.29	161437.18	162190.29	1.48	112094.64	141282.66	120342.53	122420.23	2.57	131045.47	163014.42	137102.16	138659.35	8.37
	6	122266.92	145098.32	125205.67	126177.47	1.77	121521	149500.94	127300.98	129022.2	2.88	125405.93	157138.59	132360.87	133677.31	9.05
	7	107469.11	129827.95	111163.32	112509.48	1.83	175859.51	207800.4	178601.5	179852.87	2.53	125228.91	155795.38	131661.64	132488.45	9.64
d15112	0	684939.42	818157.33	706036.09	710760.63	2.36	517426.18	641829.31	551324.46	556301.82	2.77	551900.43	689325.24	579873.02	587463.43	11.03
	1	644759.99	757962.56	660069.86	661576.26	1.37	715678.26	862233.51	741880.04	747564.26	2.48	815029.39	974523.22	825699.22	827880.7	17.15
	2	425069.33	532069.62	455270.77	463233.81	1.48	688605.9	819433.07	706406.59	709748.87	2.37	828114.32	989438.22	840661.4	841950.39	10.38
	3	528177.95	640496.37	552469.54	558107.16	1.92	625623.9	754710.34	647913.18	653709.95	2.67	689450.94	845190.23	715187.91	719245.2	10.51
	4	586915.82	699363.79	601979.1	606492.23	2.13	532088.98	654870.95	559441.99	565671.15	2.82	560385.47	708766.08	595303.59	603549.84	10.32
	5	422195.61	515031.87	444984.27	447718.84	1.4	500455.25	628035.12	536043.76	544380.46	3.3	821959.4	984976.19	837274.82	840517.07	11.36
	6	518793.6	641274.92	544960.82	558818.71	2.15	497229.6	607896.12	517291.22	525496.18	2.28	715206.03	881295.2	740882.05	750795.03	11.22
	7	616918.44	740763.28	638550.89	646697.3	2.09	599776.85	735296.99	621109.86	625506.13	2.75	958278.86	1142296.19	969527.83	971665.89	9.34
nrw1379	0	64359.77	77080.18	66757.03	67158.9	1.89	66839.83	80056.26	68699.48	68964.28	2.88	64831.76	82168.37	68703.11	69637.88	9.88
	1	83410.67	97431.46	84816.46	85171.2	1.9	65103.43	79386.02	68201.09	68785.5	3.24	64967.83	81696.27	69112.59	69537.29	9.5
	2	52858.87	66568.48	56985.44	58051.2	1.6	63480.7	78491.28	67126.76	67944.31	3	73858.13	91658.2	77067.13	77991.08	9.5
	3	62341.36	74577.8	64886.71	65147.08	2.1	59273.92	73083.73	62640.15	63176.78	3.22	100592.83	120155.76	102066.7	102383.27	7.09
	4	56012.13	67591.94	58410.29	59061.24	2.38	70594.56	83899.71	72124.06	72591.8	2.63	98228.29	118173.68	99705.52	100243.06	9.94
	5	58083.8	70546.56	61117.82	61654.89	1.89	73884.17	87852.56	75518.33	76091.15	3.03	75984.21	94166.17	79689.16	80121.07	9.54
	6	52224.66	65006.49	55878.15	56509.68	2.03	64306.14	78763.67	67312.21	68012.34	2.36	79165.6	95663.16	80952.64	81340.65	9.29
	7	58402.97	70538.97	60846.77	61262.06	2.32	90554.87	106511	91776.84	92060.2	2.73	73194.55	90341.07	76232.12	76866.8	8.46

Table 4. The experimental results for dataset type 2 (tai instances)

Instances	*Init.Sol*	Our algorithm		
		best.sol	*aver.sol*	*T*
tai75a	6840.64	4922.41	4944.85	5.68
tai75b	5575.85	3878.92	3889.81	3.56
tai75c	7110.74	3938.11	3943.07	3.56
tai75d	6501.43	4940.87	4953.09	4.04
tai100a	11398.6	7905.68	7938.74	16.40
tai100b	9775.59	7250.37	7282.37	17.00
tai100c	7895.75	4882.14	4891.52	17.40
tai100d	9051.64	5503.64	5518.74	16.40
tai150a	16188.04	14400.04	14424.68	83.00
tai150b	14018.07	11557.99	11585.42	84.50
tai150c	13293.35	10012.81	10022.76	81.40
tai150d	13001.42	10033.97	10064.00	82.30

Table 5. The experimental results for dataset type 2 (CMT instances)

Instances	*n*	*k*	*MD*	*init.sol*	*best.sol*	*aver.sol*	gap_2	*T*
CMT6	50	6	200	2986.76	2011.86	2155.06	32.64	0.93
CMT7	75	11	160	3410	2332.44	2530.69	31.60	5.52
CMT8	100	9	230	6325.56	4360.79	4698.06	31.06	18.69
CMT9	150	14	200	7873.37	5369.92	5920.42	31.80	82.90
CMT10	199	18	200	9299.84	6873.20	7418.16	26.09	323.11
CMT13	120	11	720	9056.03	6925.13	7204.73	23.53	28.20
CMT14	100	11	1040	4368.63	3937.02	4259.01	9.88	16.50
Aver							26.66	67.98