An Efficient ACO+RVND for Solving the Resource-Constrained Deliveryman Problem

Ha-Bang Ban
School of Information and Communication Technology
Hanoi University of Science and Technology
Email: BangBH@soict.hust.edu.vn; Bang.BanHa@hust.edu.vn

Communication: received 20 December 2019, revised 21 February 2020, accepted 17 April 2020
Digital Object Identifier: 10.32913/mic-ict-research.v2020.n1.911

Abstract: In this paper, Resource-Constrained Deliveryman Problem (RCDMP) is introduced. The RCDMP problem deals with finding a tour with minimum waiting time sum so that it consumes not more than R_{max} units of resources, where R_{max} is some constant. Recently, an algorithm developed in a trajectory-based metaheuristic has been proposed. Since the search space of the problem is a combinatorial explosion, the trajectory-based sequential can only explore a subset of the search space, therefore, they easily fall into local optimal in some cases. To overcome the drawback of the current algorithms, we propose a population-based algorithm that combines an Ant Colony Algorithm (ACO), and Random Variable Neighborhood Descent (RVND). In the algorithm, the ACO explores the promising solution areas while the RVND exploits them with the hope of improving a solution. Extensive numerical experiments and comparisons with the state-of-the-art metaheuristic algorithms in the literature show that the proposed algorithm reaches better solutions in many cases.

Keywords: *RCDMP, ACO, GA, metaheuristic.*

I. INTRODUCTION

The Resource-Constrained Deliveryman Problem (RCDMP) has not been studied much in literature. Informally, the RCDMP aims to find a tour with minimum waiting time sum so that it consumes no more than R_{max} units of resources, where R_{max} is some constant. The resource consumption of a tour is the sum of the resource consumptions of its edges. In the special case, since resource-constraint is ignored, the RCDMP becomes the Deliveryman Problem (DMP) or Traveling Repairman Problem (TRP). The problem has applications in reality, e.g., disk head scheduling [1, 5, 6, 8, 17, 19].

The RCDMP is at least as hard as DMP, therefore it is a NP-hard problem. The RCDMP is formulated as followings: Given a complete graph $K_n = (V, E)$ where $V = 1, 2, \ldots n$ is a set of vertices, and E is the set of edges. For each edge $(v_i, v_j) \in E$, which connects the two vertices v_i and v_j, there exists a cost $c(v_i, v_j)$ and resource consumption $r(v_i, v_j)$. Suppose that $T = (v_1, \ldots, v_k, \ldots, v_n)$ is a tour in K_n. Denote $P(v_1, v_k)$ is the path from v_1 to v_k on the tour T and $l(P(v_1, v_k))$ is its length. The waiting time of a vertex $v_k (1 < k \leq n)$ on T is the length of the path from starting vertex v_1 to v_k:

$$w(v_k) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}).$$

The total waiting time of tour T is defined as the sum of waiting time of all vertices:

$$L(T) = \sum_{k=2}^{n} w(v_k).$$

The resource consumption of a tour is the sum of the resource consumption of its edges. Tour T must satisfy the following resource constraint:

$$\sum_{i=1}^{n-1} r(v_i, v_{i+1}) \leq R_{max}. \quad (1)$$

The RCDMP asks for a tour with minimum waiting time sum so that it consumes not more than R_{max} units of resource.

The RCDMP is a NP-hard problem; thus, a metaheuristic algorithm is a suitable approach for it in short computation time. Previously, the only algorithm was developed in a trajectory-based metaheuristic approach to solve the RCDMP [4]. This algorithm starts with a single initial solution and, at each step of the search, the current solution is replaced by another solution found in its neighborhood. However, this algorithm may implement a strong intensification strategy. However, its diversification may not be good enough. As a result, it gets stuck in local optima in some cases. Another approach is population-based which performs a search with multiple promising solutions, thus, its exploring search space is extended. As a result, the chance to obtain better solutions is higher.

In this paper, we propose a population-based algorithm that combines an Ant Colony algorithm (ACO) [7], and Random Variable Neighborhood Descent (RVND) [16]. The two important features in our algorithm are that:

- Two kinds of ants coexist in the ACO. In a computational experiment, the researchers [10] performed the feeding behavior by using intelligent ants, which can trail the pheromone exactly, and dull ants which cannot trail the pheromone. From results, the ant group that includes the dull ants can obtain more foods than the group consisting of only the intelligent ants. The dull ants are regarded as having a task that finds the new food sources by dawdle. It means that the coexistence of the intelligent and dull ant improves the effectiveness of the feeding behavior.
- To be successful, a search algorithm needs to establish a good ratio between exploration and exploitation. In our algorithm, the ACO explores the promising solution areas while the RVND exploits them with the hope of improving a solution. Extensive numerical experiments and comparisons with the state-of-the-art metaheuristic algorithms on 320 instances show that the proposed algorithm reaches better solutions in 303 cases.

The rest of this paper is organized as follows. Section 2 presents the proposed algorithm. Computational evaluations are reported in section 3. Sections 4, 5 discuss and conclude the paper, respectively.

II. THE PROPOSED ALGORITHM

In this paper, we propose an algorithm which utilizes the advantages of both the ACO [7], and RVND [16]. First, in the ACO algorithm [7], multiple solutions called “ants" coexist, and the ants drop pheromone on the path connecting the locations. Pheromone trails are updated depending on the behavior of the ants. The ants find a food source through paths having strong pheromone. By communicating with other ants according to the pheromone strength, the algorithm tries to find the optimal solution. Lastly, the RVND [16] is based on a simple principle of systematic switches between different neighborhoods.

A pseudocode of our algorithm is given in Algorithm 1. Our algorithm is repeated a number of times, and the best solution found is reported. In Step 1, the ACO is used to generate an initial ant population that is an input for the RVND in Step 2. In Step 2, to exploit promising solution spaces, the RVND is implemented with the best solution from the ACO. Our algorithm stops after m iterations. If the best solution has not been improved. In the remaining of this section, more details about the three steps of our algorithm are given in Algorithm 1.

Algorithm 1 ACO-RVND

Input: K_n, Sp, and τ_0 are the graph, the cost matrix, the size of ant population, and the initial amount of pheromone deposited, respectively.
Output: The best solution T^*.

```
while (The termination criterion of the ACO-RNVD is not satisfied) do
  {Step 1: The ACO step}
  Initiate pheromone;
  while (|P| < Sp) do
    {Choose the type of ant. If the value of rd is less than 80, the ant depends on the intelligent type. Otherwise, it fails into dull type}
    rd = random(100);
    T_k = T_k ∪ v_1;
    v = v_1; {v is a next vertex}
    while | T_k |≤ n do
      N_k = {v_j | v_j ∉ T_k and v_j ∈ K_n};
      v = Pick-Forward-Vertex(v, rd, N_k);
      T_k = T_k ∪ v;
    end while
    Update phernomone τ_ij according to (4), (5);
    P = P ∪ T_k; {Add T_k to the ant population P}
  end while
  {Step 2: The RVND step}
  for T ∈ P do
    T' = RVND(T);
    if (L(T') < L(T*)) and (T' is feasible)) then
      T* = T';
      improved=True;
    end if
  end for
  {Step 3: Update pheronome}
  if (improved==True) then
    Update pheronome information by (6), and (7);
  end if
end while
return T*;
```

1. Penalty on infeasible solution

Our search is not constrained to feasible solutions. During the search we penalize infeasible solutions by incorporating a penalty term. For each solution T, let $L(T)$ denote the total latency and let $V(T)$ denote the total violation of the constraint. The total latency violation $V(T)$ is computed on a tour basis with respect to the value R_{max}. Specifically, it is equal to

$$\max\{LR - R_{max}, 0\},$$

where LR is the resource consumption value in the current solution (feasible or infeasible).

Algorithm 2 Pick-Forward-Vertex(v_i, rd, N_k)

Input: v_i, rd, N_k are the current vertex, a random number, and a set of unvisited vertices, respectively.
Output: A next vertex v.

```
proSum = 0;
if (rd ≤ ant_type) then
  {The second only uses the amount of phernomone as (2)}
  for (v_j ∈ N_k) do
    proSum = proSum + pow(τ[i,j], β_τ) × pow(β[i,j], β_μ);
  end for
  for (v_j ∈ N_k) do
    p[i,j] = pow(τ[i,j],β_τ)×pow(η[i,j],β_μ) / proSum;
  end for
  rd = random(proSum);
  for (v_j ∈ N_k ) do
    wheelSum = wheelSum + p[i,j];
    if (wheelSum > rd) then
      break;
    end if
    v = v_j;
  end for
end if
{The third cannot trail the pheromone as (3)}
if (rd > ant_type) then
  Select randomly vertex v ∈ N_k;
end if
```

Algorithm 3 RVND(T)

Input: T is a tour.
Output: A new solution T.

```
Initialize the Neighborhood List NL;
while NL ≠ 0 do
  Choose a neighborhood N in NL at random
  T' ← arg min N(T); {Neighborhood search}
  if ((L(T') < L(T)) then
    T ← T'
    Update NL;
  else
    Remove N from the NL;
  end if
end while
```

Solutions are then evaluated according to the weighted fitness function $L' = L + PF \times V(T)$, where PF is the penalty factor. If the obtained solution is feasible then $LR \leq R_{max}$ and $L' = L$.

2. Step 1: ACO

Step 1 includes some small steps as follows:
- **Encoding for ants**: We simply use the natural encoding, in which an ant individual is represented as a list of n vertices $(v_1, v_2, ..., v_k, ..., v_n)$, where v_k is the $k-$th vertex to be visited. Each individual represents one possible tour.
- **Initializing parameters for the ACO**: Let τ_{k-1ij} be the amount of pheromone on the edge (v_i, v_j) so that the $k-$th ant uses to trail $(k = 1, 2, ..., Sp)$. Initially, τ_{0ij} is set to τ_0, respectively.
- **Selecting for the next vertex**: Assume that, $k-$th ant needs to be chosen to go from vertex v_i to a next vertex in an unvisited vertex set N_k. Our algorithm maintains two types of ant:
The first uses the amount of phernomone thus, its probability equation is as follows:

$$p^2_{kij} = \frac{[\tau_{k-1ij}]^{\beta_r}[\eta_{k-1ij}]^{\beta_\eta}}{\sum_{v_j \in N_k}[\tau_{k-1ij}]^{\beta_r}[\eta_{k-1ij}]^{\beta_\eta}} \quad (2)$$

The last does not use either the phernomone or genetic information, therefore its probability equation is as follows:

$$p^3_{kij} = \frac{1}{|N_k|} \quad (3)$$

where β_r, β_η, and β_g are parameters, $\eta_{kij} = \frac{1}{\rho \times c_{ij}}$ (ρ, and $|N_k|$ are the position of edge (v_i, v_j), and number of vertices in the tour, respectively). The selection of v_j is done by the roulette wheel selection [9]. The ants repeat choosing a next vertex until all vertices are visited. The detail in this step is given in Algorithm 2.
- **Updating Pheromone of $k-$ant**: After $k-$ant has completed the tour T_k, its deposited pheromone on T_k is updated. We should note that the dull ants can deposit the pheromone although they cannot trail the pheromone. The mount of pheromone which an ant deposits on T_k is calculated as follows:

$$\Delta\tau_{kij} = \begin{cases} \frac{\sigma}{L(T_k)}, & \text{if } (v_i, v_j) \in T_k \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where σ is a parameter. Since the k-ant finishes its tours, we update τ_{ij} of each edge (v_i, v_j) depending on $\Delta\tau_{kij}$ as follows:

$$\tau_{kij} = (1 - p) \times \tau_{k-1ij} + \Delta\tau_{kij} \quad (5)$$

where $p \in (0, 1)$ is the pheromone trail decay coefficient.

3. Step 2: RVND

To be successful, a search algorithm needs to establish a good ratio between exploration and exploitation. In the proposed algorithm, the ACO is also combined with the RVND to keep a balance between exploration and

exploitation. In this combination, the ACO explores the promising solution spaces while the RVND bases on a simple principle of systematically switches between different neighborhoods to exploit these spaces. In the RVND step, several neighborhoods are used such as remove-insert, reinsertion, swap-adjacent, swap, 2-opt, and or-opt in [16]. In THE RCDMP, by using the known cost of the current solution, this operation can be done in constant time [4]. Therefore, the running time of exploring the neighborhoods can be sped up. Six neighborhoods used in the step are described as followings: 1) **Swap-adjacent**: It swaps each pair of adjacent vertices. The complexity of exploring the neighborhood is $O(n)$; 2) **Remove-insert** moves each vertex in the solution at the end of it. The complexity of exploring the neighborhood is $O(n)$; 3) **Reinsertion**: One vertex is relocated to another position of the tour. The complexity of exploring the neighborhood is $O(n)$; 4) **Swap**: It tries to swap the positions of each pair of vertices in the single tour T. The complexity of exploring the neighborhood is $O(n^2)$; 5) **2-opt**: It removes each pair of edges from the tour and reconnects them. The complexity of exploring the neighborhood is $O(n^2)$; 6) **Or-opt**: Three adjacent vertices are reallocated to another position of the tour. The complexity of exploring the neighborhood is $O(n^2)$. A pseudocode of the RVND algorithm is given in Algorithm 3.

4. Step 3: Update pheromone

If the new best solution is found, we use it to update pheromone value. The pheromone information $\Delta\tau^*_{ij}$ bequeathed to the ACO in the next iteration is calculated as follows:

$$\Delta\tau^*_{ij} = \begin{cases} \dfrac{\sigma}{L(T^*)}, & \text{if } (v_i, v_j) \in T^* \\ 0, & \text{otherwise;} \end{cases} \quad (6)$$

where σ, $L(T^*)$ are parameter, and the cost of T^*, respectively. Therefore, the pheromone information on each edge (v_i, v_j) is updated by $\Delta\tau^*_{ij}$ as follows:

$$\tau_{ij} = \tau_{ij} + \Delta\tau^*_{ij} \quad (7)$$

III. 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 8 GB bytes RAM memory.

1. Benchmark Instances

The numerical analysis was performed on a set of benchmarks described in [4, 19, 23]. In our experiments, two types of dataset are selected as followings:

- The RCDMP dataset in [4]: Cost matrix elements, c_{ij}, are independent and uniformly chosen random integers in the range [0, 200]. Resource matrix elements, r_{ij}, are independent and uniformly chosen integers in the range [0, 200 - c_{ij}]. The cost matrix $C = [c_{ij}]$ and $R = [r_{ij}]$ are generally symmetric and metric. Maximum resource usage, R_{max}, is computed using the following formula:

$$R_{max} = \lceil \alpha \times \sum_{i \in V} \sum_{j \in V} r_{ij} x^c_{ij} + (1 - \alpha) \times \sum_{i \in V} \sum_{j \in V} r_{ij} x^r_{ij} \rceil$$

  In the above formula, x^c_{ij} represent the optimal solution of the unconstrained DMP with the cost matrix c_{ij}. Similarly, x^r_{ij} represent the optimal solution of the problem

$$\sum_{i=1}^{n-1} r(v_i, v_{i+1}) \rightarrow \min,$$

  where the cost matrix is defined by the matrix r_{ij}. For the problem, we choose the Concorde tool [1] to solve exactly the problem. In the equation, α is a parameter, that provides means for controlling tightness of the resource constraint. When $\alpha = 0$, the resource constraint is very tight. On the other hand, when $\alpha = 1$, the resource constraint has no effect on the optimal solution. We choose $\alpha = 0, 0.5, 0.75$ and 1 and vary the size of the instances between 30 to 100 vertices with the different values of α to create 320 instances. For unconstrained DMP with a small number of instances ($n = 30, 40, 50$), we use the exact algorithm in [1, 2] to solve. Currently, there is no exact algorithm in the literature that can solve the problem with up to 100 vertices. Therefore, we use the metaheuristic in [3] to obtain near-optimal solutions. All instances are available in the online supplement to this paper at [2].
- Salehipour et al.'s dataset in [19]: Five of these sets are generated by Salehipour et al., where each of them is composed of 20 instances with 10, 20, 50, 100, and 200 customers, respectively. In total, there are 100 instances.
- TSPLIB in [23]: Some instances, that are from 70 to 150 vertices are selected. The optimal solutions to them are extracted from [1]. In total, there are 11 instances.

The selection of pheromone-related parameters are conducted in preliminary experiments. The parameter *ant_type* is set to 80 because it has been reported that about 20 percent of the dull ant and 80 percent of the intelligent ant live in the real ant's world in [10]. For the

[1] http://www.math.princeton.edu/tsp/concorde.html
[2] https://drive.google.com/drive/u/2/folders/1M_zAeuJXpOuVlnmdmwg6pjDSn4ctA9X3?ths=true

Table 2. The experiment results for the RCDMP-instances (n = 30)

Instances	α = 0					α = 0.5					α = 0.75					α = 1				
	TS	Our algorithm				TS	Our algorithm				TS	Our algorithm				TS	Our algorithm			
	Best.Sol	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*
test-1	6654	**6259**	6259	-5.94	0.13	6611	**6470**	6470	-2.13	0.13	6391	**6351**	6351	-0.63	0.12	6391	**6331**	6331	-0.94	0.10
test-2	7130	**6854**	6854	-3.87	0.11	7289	**6983**	6983	-4.20	0.12	7050	**6710**	6710	-4.82	0.15	6654	6903	6903	3.74	0.11
test-3	6546	**6042**	6042	-7.70	0.14	6441	**5939**	5939	-7.79	0.14	6430	**5958**	5958	-7.34	0.12	6381	**6060**	6060	-5.03	0.14
test-4	6866	**6338**	6338	-7.69	0.11	6879	**6260**	6260	-9.00	0.13	6715	**6407**	6407	-4.59	0.11	6654	**6336**	6336	-4.78	0.10
test-5	6619	**6124**	6124	-7.48	0.13	6551	**6111**	6111	-6.72	0.12	6674	**6021**	6021	-9.78	0.15	6215	**6246**	6246	0.50	0.15
test-6	6652	**6525**	6525	-1.91	0.11	6780	**6448**	6448	-4.90	0.15	6690	**6520**	6520	-2.54	0.15	6654	**6435**	6435	-3.29	0.14
test-7	7189	**6804**	6804	-5.36	0.12	7269	**6918**	6918	-4.83	0.14	7457	**6710**	6710	-10.02	0.12	6654	6815	6815	2.42	0.12
test-8	5888	**5569**	5569	-5.42	0.13	5956	**5713**	5713	-4.08	0.13	5830	**5720**	5720	-1.89	0.11	5830	**5638**	5638	-3.29	0.13
test-9	6063	**5862**	5862	-3.32	0.14	6228	**5718**	5718	-8.19	0.13	6051	**5574**	5574	-7.88	0.11	5878	**5867**	5867	-0.19	0.11
test-10	6475	**6047**	6047	-6.61	0.10	6263	**6080**	6080	-2.92	0.13	6257	**6201**	6201	-0.89	0.12	6257	**6207**	6207	-0.80	0.12
test-11	7424	**6952**	6952	-6.36	0.15	7499	**6871**	6871	-8.37	0.11	7590	**7116**	7116	-6.25	0.13	6654	**6769**	6769	1.73	0.15
test-12	6371	**5796**	5796	-9.03	0.14	6110	**5807**	5807	-4.96	0.12	6357	**5936**	5936	-6.62	0.11	6110	**5909**	5909	-3.29	0.13
test-13	6029	**5671**	5671	-5.94	0.12	6083	**5736**	5736	-5.70	0.12	6134	**5680**	5680	-7.40	0.13	5885	**5739**	5739	-2.48	0.13
test-14	6160	**5613**	5613	-8.88	0.12	5545	**5678**	5678	2.40	0.11	5960	**5613**	5613	-5.82	0.14	5545	**5606**	5606	1.10	0.11
test-15	6387	**5810**	5810	-9.03	0.12	6237	**5832**	5832	-6.49	0.14	6013	**5993**	5993	-0.33	0.11	6013	6040	6040	0.45	0.12
test-16	6772	**6532**	6532	-3.54	0.12	6818	**6409**	6409	-6.00	0.11	6794	**6468**	6468	-4.80	0.11	6654	**6348**	6348	-4.60	0.13
test-17	6591	**6509**	6509	-1.24	0.13	7069	**6546**	6546	-7.40	0.11	6988	**6699**	6699	-4.14	0.11	6654	6804	6804	2.25	0.13
test-18	6943	**6460**	6460	-6.96	0.13	6856	**6657**	6657	-2.90	0.11	6736	**6520**	6520	-3.21	0.12	6654	**6576**	6576	-1.17	0.12
test-19	7622	**7258**	7258	-4.78	0.14	7289	**6942**	6942	-4.76	0.11	7354	**7194**	7194	-2.18	0.12	6654	**7076**	7076	6.34	0.12
test-20	6472	**5848**	5848	-9.64	0.14	6352	**5985**	5985	-5.78	0.12	6409	**5947**	5947	-7.21	0.13	6352	**5938**	5938	-6.52	0.15
Aver				-6.03	0.13				-5.24	0.12				-4.92	0.12				-0.89	0.13

Improved solutions are highlighted in boldface

Table 3. The experiment results for the RCDMP-instances (n = 40)

Instances	α = 0					α = 0.5					α = 0.75					α = 1				
	TS	Our algorithm				TS	Our algorithm				TS	Our algorithm				TS	Our algorithm			
	Best.Sol	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*
test-1	10745	**10606**	10606	-1.29	0.25	10764	**10622**	10622	-1.32	0.20	10851	**10525**	10525	-3.00	0.21	10745	**10508**	10508	-2.21	0.22
test-2	9934	**9539**	9539	-3.98	0.25	9925	**9324**	9324	-6.06	0.23	9867	**9491**	9491	-3.81	0.21	9605	**9252**	9252	-3.68	0.20
test-3	9800	**9400**	9400	-4.08	0.20	10226	**9572**	9572	-6.40	0.24	9715	**9352**	9352	-3.74	0.23	9715	**9238**	9238	-4.91	0.23
test-4	11222	**10317**	10317	-8.06	0.24	11054	**10167**	10167	-8.02	0.23	11002	**9886**	9886	-10.14	0.21	10745	**10248**	10248	-4.63	0.22
test-5	10209	**9413**	9413	-7.80	0.21	9743	**9423**	9423	-3.28	0.21	10118	**9310**	9310	-7.99	0.24	9743	**9607**	9607	-1.40	0.23
test-6	10117	**9272**	9272	-8.35	0.22	9684	**9412**	9412	-2.81	0.22	9892	**9243**	9243	-6.56	0.25	9684	**9276**	9276	-4.21	0.23
test-7	10182	**8824**	8824	-13.34	0.23	9899	**9086**	9086	-8.21	0.22	10333	**9219**	9219	-10.78	0.24	9629	**9262**	9262	-3.81	0.23
test-8	9456	**8787**	8787	-7.07	0.25	9443	**8874**	8874	-6.03	0.25	9267	**9034**	9034	-2.51	0.22	9267	**8822**	8822	-4.80	0.20
test-9	10978	**10470**	10470	-4.63	0.22	10980	**10491**	10491	-4.45	0.21	10948	**10491**	10491	-4.17	0.23	10745	**10317**	10317	-3.98	0.20
test-10	9568	**8884**	8884	-7.15	0.25	9809	**8847**	8847	-9.81	0.24	9553	**9043**	9043	-5.34	0.21	9553	**8780**	8780	-8.09	0.22
test-11	10456	**9773**	9773	-6.53	0.22	10207	**9811**	9811	-3.88	0.23	10802	**9478**	9478	-12.26	0.25	9621	9610	9610	-0.11	0.23
test-12	9728	**9123**	9123	-6.22	0.24	9932	**9186**	9186	-7.51	0.22	9860	**9245**	9245	-6.24	0.24	9735	**9361**	9361	-3.84	0.23
test-13	10829	**10246**	10246	-5.38	0.23	10407	**10234**	10234	-1.66	0.21	10232	**10190**	10190	-0.41	0.24	10232	**9948**	9948	-2.78	0.22
test-14	10527	**9779**	9779	-7.11	0.23	10368	**9882**	9882	-4.69	0.22	10203	**10057**	10057	-1.43	0.21	10111	10165	10165	0.53	0.24
test-15	10146	**9513**	9513	-6.24	0.23	10288	**9668**	9668	-6.03	0.22	10310	**9434**	9434	-8.50	0.23	10120	**9288**	9288	-8.22	0.24
test-16	10624	**10440**	10440	-1.73	0.23	10872	**10305**	10305	-5.22	0.21	10704	**10109**	10109	-5.56	0.20	10704	**10201**	10201	-4.70	0.25
test-17	10653	**10005**	10005	-6.08	0.21	10102	**9487**	9487	-6.09	0.23	10654	**10075**	10075	-5.43	0.22	10102	**9910**	9910	-1.90	0.23
test-18	10140	**9546**	9546	-5.86	0.21	9651	**9174**	9174	-4.94	0.21	10047	**9210**	9210	-8.33	0.22	9651	**9419**	9419	-2.40	0.22
test-19	9490	**8956**	8956	-5.63	0.25	10009	**9354**	9354	-6.54	0.22	9714	**9410**	9410	-3.13	0.21	9714	**9339**	9339	-3.86	0.21
test-20	9500	**8578**	8578	-9.71	0.21	9415	**8589**	8589	-8.77	0.23	9183	**8478**	8478	-7.68	0.21	9183	**8489**	8489	-7.56	0.23
Aver				-6.31	0.23				-5.59	0.22				-5.85	0.22				-3.83	0.22

Improved solutions are highlighted in boldface

other parameters, the best configuration is presented as follows: As finding the best configuration by running all instances would be computationally too expensive, we implement our numerical analysis on some selected instances.

Table 4. The experiment results for the RCDMP-instances (n = 50)

Instances	α = 0					α = 0.5					α = 0.75					α = 1				
	TS	Our algorithm				TS	Our algorithm				TS	Our algorithm				TS	Our algorithm			
	Best.Sol	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*
test-1	13691	**13237**	13350.1	-3.32	1.13	14069	**13232**	13453.8	-5.95	0.98	14055	**13392**	13534.8	-4.72	0.95	13691	**13107**	13364.9	-4.27	1.09
test-2	13419	**11881**	12296.6	-11.46	1.03	13369	**12292**	12509.3	-8.06	0.97	12785	**12577**	12688.1	-1.63	1.12	12785	**12285**	12654.3	-3.91	1.10
test-3	13740	**12911**	13102.2	-6.03	0.93	14210	**13020**	13230.6	-8.37	1.10	13521	**12781**	13084.4	-5.47	1.04	13521	**13032**	13161.5	-3.62	1.09
test-4	14848	**14529**	14616.9	-2.15	0.98	15178	**14016**	14510.3	-7.66	1.15	15261	**14380**	14585.3	-5.77	0.95	13691	**13691**	13691	0.00	1.18
test-5	15539	**14227**	14503.1	-8.44	0.95	14798	**13928**	14319.8	-5.88	1.00	14770	**14466**	14591.5	-2.06	1.00	13691	14243	14372	4.03	0.96
test-6	14925	**13778**	13959.1	-7.69	0.98	14496	**13956**	14132.7	-3.73	1.13	14776	**14069**	14262.7	-4.78	1.08	13691	**13691**	13691	0.00	1.11
test-7	15657	**13425**	14141.6	-14.26	1.03	15658	**14144**	14480.3	-9.67	1.10	15087	**13917**	14113.4	-7.76	0.96	13691	**13691**	13691	0.00	0.97
test-8	15966	**14819**	15025.2	-7.18	1.06	15555	**14758**	14873.9	-5.12	0.90	15902	**14646**	14829.3	-7.90	1.12	13691	**14765**	14992.1	7.84	0.94
test-9	15815	**15446**	15599.2	-2.33	1.04	16105	**14974**	15187.9	-7.02	1.08	16176	**15003**	15159.4	-7.25	0.97	13691	**13691**	13691	0.00	1.08
test-10	12819	**11956**	12120.8	-6.73	1.16	12686	**12079**	12283.5	-4.78	1.02	12800	**11865**	12232.4	-7.30	1.18	12686	**11961**	12177.1	-5.71	1.04
test-11	15237	**13977**	14221.7	-8.27	1.06	14370	**14125**	14321.8	-1.70	1.17	14670	**13780**	14137.8	-6.07	0.98	13691	**13620**	13941.3	-0.52	1.04
test-12	15204	**14241**	14517.8	-6.33	1.18	15275	**14134**	14464.5	-7.47	0.90	15399	**14093**	14598.6	-8.48	1.13	13691	14417	14705.4	5.30	1.10
test-13	14119	**12908**	13134.9	-8.58	1.09	14347	**12900**	13084.7	-10.09	1.04	14406	**13025**	13195.3	-9.59	0.96	13691	**13113**	13234.6	-4.22	1.13
test-14	14551	**13363**	13659.7	-8.16	1.19	14939	**13070**	13566.8	-12.51	1.03	14721	**13750**	13989.1	-6.60	0.99	13691	**13358**	13520.7	-2.43	1.01
test-15	14657	**13939**	14135.1	-4.90	0.97	14845	**13964**	14136.4	-5.93	1.04	14080	**13706**	14172.4	-2.66	0.93	13691	13748	14181	0.42	1.10
test-16	15053	**14065**	14267.6	-6.56	1.10	15165	**14186**	14288.4	-6.46	1.13	15312	**14257**	14360.6	-6.89	1.07	13691	14008	14220.2	2.32	1.02
test-17	14587	**13213**	13503.4	-9.42	0.99	14208	**13072**	13580.2	-8.00	1.00	14413	**13561**	13745.3	-5.91	1.11	13691	**13247**	13558.4	-3.24	1.15
test-18	13652	**12727**	12925.8	-6.78	1.10	13844	**13035**	13187.9	-5.84	1.14	14403	**12862**	13044.3	-10.70	1.06	13691	**12852**	13060	-6.13	1.15
test-19	15883	**14600**	14744.2	-8.08	1.11	15521	**14490**	14840.5	-6.64	1.04	15902	**14808**	15000.2	-6.88	1.03	13691	14905	14995.9	8.87	0.98
test-20	14771	**13700**	13918.6	-7.25	0.92	15259	**13815**	14111.2	-9.46	0.91	14649	**13885**	14134.4	-5.22	1.09	13691	13939	14109.7	1.81	1.08
Aver				-7.20	1.05				-7.02	1.04				-6.18	1.04				-0.17	1.07

Improved solutions are highlighted in boldface

Table 5. The experiment results for the RCDMP-instances (n = 100)

Instances	α = 0					α = 0.5					α = 0.75					α = 1				
	TS	Our algorithm				TS	Our algorithm				TS	Our algorithm				TS	Our algorithm			
	Best.Sol	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*	*Best.Sol*	*Best.Sol*	*Aver.Sol*	gap_2	*T*
test-1	40496	**39988**	40567.1	-1.25	11.75	39729	**39585**	39950.3	-0.36	11.21	40067	**39781**	40245.2	-0.71	11.04	38855	**38668**	39105.9	-0.48	12.47
test-2	39441	**39207**	38988.3	-0.59	11.62	38249	**38127**	38988.3	-0.32	11.35	39585	**39263**	38988.3	-0.81	10.45	38745	**38686**	38988.3	-0.15	11.29
test-3	40045	**40029**	40339.6	-0.04	12.61	39804	**39690**	40132.8	-0.29	11.10	40134	**40037**	40310.3	-0.24	11.76	40323	**40239**	40451.8	-0.21	12.66
test-4	38882	**38762**	39149.6	-0.31	10.79	38402	**38395**	38879.7	-0.02	12.29	38067	**37977**	38470.2	-0.24	10.79	39190	**39182**	39490.1	-0.02	11.17
test-5	40414	**40404**	40552.2	-0.02	10.95	40752	**40726**	41003.9	-0.06	11.88	40885	**40741**	41171.2	-0.35	10.13	40212	**40142**	40408.9	-0.17	12.31
test-6	38338	**38271**	38543.2	-0.17	10.36	39100	**38744**	39165.2	-0.91	12.32	38954	**38916**	39298.9	-0.10	12.26	39257	**39203**	39332.6	-0.14	11.19
test-7	38729	**38243**	38954.3	-1.25	12.82	38005	**37984**	38536.2	-0.06	12.80	37812	**37729**	37937.1	-0.22	10.73	38567	**38492**	38775.9	-0.19	12.43
test-8	41141	**41056**	41281	-0.21	11.94	41087	**41076**	41244.2	-0.03	12.92	40663	**40627**	40985.1	-0.09	11.33	39661	**39585**	39837.9	-0.19	12.27
test-9	38705	**38311**	39190.3	-1.02	11.44	38308	**38106**	38804	-0.53	10.58	39824	**39602**	39909.4	-0.56	12.06	38982	**38954**	39303.2	-0.07	11.13
test-10	38219	**38205**	38519.9	-0.04	11.92	38888	**38883**	39614.1	-0.01	10.42	39188	**39173**	39280.5	-0.04	11.08	38318	**38226**	38453.4	-0.24	10.65
test-11	39112	**39009**	39234.5	-0.26	11.63	39392	**39366**	39860.7	-0.07	12.09	39299	**39279**	39536.5	-0.05	12.21	38921	**38763**	39189	-0.41	12.37
test-12	38519	**38447**	38768	-0.19	11.94	38095	**37977**	38160.3	-0.31	10.28	38102	**38007**	38279.7	-0.25	11.18	38076	**38038**	38196.4	-0.10	12.85
test-13	40809	**40765**	41150.4	-0.11	11.63	39986	**39738**	40480.1	-0.62	11.58	40233	**40219**	40738.8	-0.03	12.05	40670	**40648**	40888.2	-0.05	10.98
test-14	40199	**39993**	40849	-0.51	12.16	40745	**40727**	41192.1	-0.04	11.59	39386	**39330**	40120.7	-0.14	12.11	40294	**40157**	40518.6	-0.34	12.01
test-15	39754	**39728**	40062.3	-0.07	11.57	39912	**39876**	39990.7	-0.09	12.58	39304	**39288**	39692.7	-0.04	11.33	39026	**38987**	39362.1	-0.10	11.32
test-16	40166	**40061**	40354.5	-0.26	12.98	40352	**40211**	40609.3	-0.35	11.45	39969	**39911**	40193.4	-0.15	10.06	40932	**40897**	41023.9	-0.09	12.50
test-17	39363	**39289**	39682.5	-0.19	10.66	38991	**38885**	39433.9	-0.27	11.18	39531	**39505**	39932.4	-0.07	10.99	40096	**40017**	40178.3	-0.20	12.31
test-18	40212	**40169**	40362.5	-0.11	10.32	39018	**38895**	39129.3	-0.32	12.01	40031	**39832**	40099.3	-0.50	11.27	40089	**40079**	40532	-0.02	10.50
test-19	39875	**39861**	40111.9	-0.04	10.33	39420	**39137**	39958.2	-0.72	12.22	39675	**39349**	39914	-0.82	10.81	39091	**39067**	39462.2	-0.06	12.59
test-20	38811	**38810**	38910.7	0.00	10.19	38204	**38086**	38291.2	-0.31	11.56	39137	**39126**	39230.6	-0.03	10.59	37869	**37864**	38342	-0.01	12.97
Aver				-0.33	11.48				-0.28	11.67				-0.27	11.21				-0.16	11.90

Improved solutions are highlighted in boldface

Table 6. The average experimental results for RCDMP-instances

Instance	α = 0		α = 0.5		α = 0.75		α = 1	
	gap_2 [%]	T	gap_2 [%]	T	gap_2 [%]	T	gap_2 [%]	T
30	-6.03	0.13	-5.24	0.12	-4.92	0.12	-0.89	0.13
40	-6.31	0.23	-5.59	0.22	-5.85	0.22	-3.83	0.22
50	-7.20	1.05	-7.02	1.04	-6.18	1.04	-0.17	1.07
100	-0.33	11.48	-0.28	11.67	-0.27	11.21	-0.16	11.90

Table 7. The experimental results for DMP-instances

Instances	GRASP– VND		GRASP–VNS		GVNS-1		GVNS-2		TS-RVNS		Our algorithm	
	gap_1 [%]	T	gap_1 [%]	T	gap_1 [%]	T	gap_1 [%]	T	gap_1 [%]	T	gap_1 [%]	T
10	33.04	0.00	33.04	0.00	33.04	0.07	33.04	0.23	33.04	0.00	**33.04**	0.10
20	39.63	0.00	40.34	0.04	39.21	0.41	39.29	0.87	39.29	0.07	**39.29**	0.21
50	45.30	0.00	47.20	3.54	43.90	2.41	43.97	4.36	43.97	0.65	**43.97**	1.12
100	43.26	1.11	44.28	103.92	40.79	25.8	40.82	38.4	40.82	7.85	**40.82**	13.52
200	43.16	3.75	38.77	3995.0	37.95	124.3	38.14	93.8	38.14	78.08	**38.14**	121.21
500	46.11	604.89	49.50	1524.09	39.41	414.05	39.54	421.72	-	-	41.94	207.94
1000	45.99	3876.51	42.35	9311.21	41.51	942.47	41.28	921.52	-	-	**41.28**	2079.67

Improved gap is highlighted in boldface

Table 8. The experimental results for the instances in TSPLIB

Instances	*UB*	Our algorithm		
		Best.Sol	*Aver.Sol*	*T*
dantzig42	12528*	**12528**	12528	1.16
att48	209320*	**209320**	209320	1.40
eil51	10178*	**10178**	10178	1.70
berlin52	143721*	**143721**	143721	1.20
st70	20557*	**20557**	20557	4.21
KroA100	983128*	**983128**	983128	14.52
KroC100	961324*	**961324**	961324	12.41
KroD100	976965*	**976965**	976965	13.31
Eil101	27513	**27513**	27513	12.41
Pr124	3154346	**3154346**	3154346	16.31
Lin105	603910	**603910**	603910	15.22

* the optimal values

Table 9. Comparison of the optimal solution for the DMP with the best-found CRDMP-solution using the CRDMP objective function

instances	α = 0		α = 0.5		α = 0.75		α = 1	
	%diff	feasibility	%diff	feasibility	%diff	feasibility	%diff	feasibility
test-x-30	4.78	No	4.79	No	4.98	No	5.36	No
test-x-40	6.24	No	6.18	No	6.17	No	6.10	No

This configuration determined in multiple combinations are tested and the one who has presented the best solution is chosen. In Table 1, we define a range for each of the five parameters that yields 1200 different parameter combinations and run the algorithm for some selected instances of these combinations. Each run is limited to 1000 iterations but was stopped when the best-known solution is found. This exhaustive search for the best parameter combinations is useful as a benchmark for evaluating the algorithm. Looking at the parameter combinations, we find the following settings so that our algorithm obtains the best solutions: $\beta_r = 5, \beta_\eta = 5, \alpha = 1$, $p = 1$, and $\tau_0 = 10$. Lastly, the last aspect to discuss is the stop criterium of the algorithm. The parameter m is set to 10 to balance between

TABLE I
THE VARIABLE PARAMETERS

Parameter	Value Range
β_r	$0 \leq \beta_r \leq 10$, incremented by 0.25
α	$0.5 \leq \alpha \leq 1.5$, incremented by 0.25
β_η	$0 \leq \beta_\eta \leq 10$, incremented by 0.25
τ_0	$0 \leq \tau_0 \leq 20$, incremented by 5
p	$0.3 \leq p \leq 1$, incremented by 0.2

computation time and efficiency. This parameter setting has thus been used in the following experiments.

2. Bounds

To evaluate the efficiency of the metaheuristic algorithm, its solution can be compared to 1) the optimal solution (*OPT*); and 2) a good upper bound (*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 ($gap_1[\%]$), and upper bound ($gap_2[\%]$) in percent respectively as follows:

$$gap_1[\%] = \frac{Best.Sol - LB}{LB} \times 100\% \quad (8)$$

$$gap_2[\%] = \frac{Best.Sol - UB}{UB} \times 100\% \quad (9)$$

We choose several state-of-the-art metaheuristic algorithms for RCDMP, and DMP [4, 17, 19] as a baseline in our research. Note that: The value of *LB* is described in [17] while the value of *UB* is given by the algorithm in [4].

3. Results and Comparative Analysis

The tables present *Best.Sol*, *Aver.Sol*, and *T*, respectively. Tables from 2 to 5 compare the results of our algorithm with the solutions obtained by Ban et al. [4] for the RCDMP-instances. Column TS in Tables 2 to 5 corresponds to the best solution of Tabu Search [4]. Column α is corresponding to the results with different resource constraint values. The results in Table 6, which are the average values calculated from Table 2 to 5. Tables 7 to 8 compare our algorithm with the state-of-the-art metaheusitics in terms of gap_2 for DMP-instances. Table 9 compares the optimal solutions with the best-found CRDMP-solutions using the CRDMP objective function. For DMP-instances, the optimal solutions are extracted from the exact algorithm in [2].

Tables 2 to 5 compare the results obtained by Ban et al. [4] to our algorithm for 320 instances. The results show that our algorithm reaches better solutions at a reasonable amount of time. Regarding the average gap_2 in Table 6, our algorithm quality improves 7.20% compared with Ban et al.'s algorithm [4]. The improvement is significant since it can be observed that our algorithm is capable of finding the new best solutions for the 303 RCDMP-instances. Note that: the found-best solutions is highlighted in bold line.

Table 7 compares the results of our algorithm with the lower bounds obtained from [19] with 100, 200, 500, and 1000 vertices for DMP. Our algorithm gives much better solutions than those of GVNS-1 [17], GRASP-VND and GRASP-VNS [19] for all instances. In comparison with GVNS-2 [17], and TS [3], for most cases, our algorithm gives the same results.

Table 8 presents the average results of our algorithm on instances with up to 150 vertices selected from the TSPLIB [23]. These solutions are compared with optimal values, found by Abeledo et al. [1]. It is noteworthy that our algorithm obtains the optimal solutions with up to 100 vertices such as KroA100, KroB100, KroC100, and KroD100 at reasonable amount of time. The experimental results show that the proposed algorithm can be applied well to the DMP (note that: the DMP is another variant of the RCDMP when resource limit is ignored), though it is not designed to solve it. Therefore, our algorithm can easily adapt to similar variants of the problem.

Table 9 shows that the optimal solutions for the DMP are generally not good solutions to the CRDMP on the same instance. Specifically, our algorithm is better than the optimal solution for the DMP using the CRDMP objective function since the difference between their objective values is up to 6.24%. Although the difference is not large, all these optimal solutions for the DMP are not feasible solutions for the CRDMP. Therefore, the methods designed for the DMP may not be adapted easily to solve the CRDMP. Developing an algorithm for the CRDMP is very necessary.

The average scaled running time of our algorithm is better than those of A. Salehipour et al.'s [19], and grow quite moderate with the one of Mladenovic et al.'s [17] and Ban et al.'s algorithm [4].

IV. DICUSSIONS

In general, metaheuristic is a suitable approach for solving the NP-hard problems in short computation time. Recently, a metaheuristic for solving the problem has been proposed. The algorithm is developed in a sequential heuristic approach which starts with a single initial solution, and at each step of the search, the current solution is replaced by another solution found in its neighborhood. Since the search space of the problem is a combinatorial explosion, these algorithms only explore a subset of the search space. Therefore, they can easily fall into the local optimal in some cases. Our hybrid algorithm that brings together the components of the ACO, and RVND can overcome this

drawback by searching from multiple promising solution areas, thus, its exploring search space is extended. As a result, the chances of finding better solutions are higher. To summarize, our scheme includes new features as follows:

- Two kinds of ants coexist in the ACO. The ant group that includes the dull ants can obtain more foods than the group consisting of only the intelligent ants. It is suitable in the real ant's world.
- Every search algorithm needs to address the exploration and exploitation of a search space. Exploration is the process of visiting entirely new regions of a search space, whilst exploitation is the process of visiting those regions of a search space within the neighborhood of previously visited points. To be successful, a search algorithm needs to establish a good ratio between exploration and exploitation. The ACO is also combined with the RVND to keep a balance between exploration and exploitation. In this combination, the ACO allows us to explore the promising solution spaces while the RVND based on a simple principle of systematic switches between different neighborhoods to exploit in these spaces.

Our experiments not only prove these expectations but also indicate that this approach finds the new best known solutions in the 303 instances in comparison with the-state-of-art metaheuristic. Moreover, our experiments show that the methods designed for the DMP [3, 17, 19] may not be adapted easily to solve the CRDMP since the optimal solutions for the DMP are infeasible solutions for the CRDMP. Therefore, it is important for developing a metaheuristic for the CRDMP .

We discuss more about some optimization techniques based on swarm intelligence. There are many optimization techniques such as Genetic Algorithm [9], Particle Swarm Optimization (PSO) [12], Artificial Bee Colony (ABC) [11], and Cuckoo Search (CS) [20], etc. In fact, no optimization technique is better than the others in all cases. However, in the paper, the ACO algorithm is used to solve the problem because: 1) the field of ACO has attracted a large number of researchers, and nowadays a large number of research results of both experimental and theoretical nature exists; 2) Its efficiency is demonstrated through solving the other variants of problem (Traveling Salesman Problem (TSP) [22], Vehicle Routing Problem (VRP) [18], ...); 3) it is very understandable and fast to implement.

V. CONCLUSIONS

In this paper, we propose the metaheuristic algorithm which is mainly based on the principles of the ACO, and RVND to solve the problem. Extensive numerical experiments on benchmark instances show that on average, our algorithm finds the new best solutions in 303 instances. For close variants of the problem, our algorithm can reach the optimal solutions for the problems with 100 vertices in a reasonable amount of time. When the problem size is too large, it takes longer time to operate the whole algorithm, lowering the efficiency of finding the optimum. In these cases, to reduce the running time, parallelizing ACO [14] and divide and conquer ACO [13] can be applied to large-scale instances. This is a topic for future research.

ACKNOWLEDGEMENT

This research is funded by Hanoi University of Science and Technology (HUST) under grant number T2017-PC-082.

REFERENCES

[1] 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.

[2] H.B. Ban, K. Nguyen, M.C. Ngo, and D.N. Nguyen, "An Efficient Exact Algorithm for Minimum Latency Problem", *J. PI*, No. 10, 2013, pp. 1-8.

[3] H.B. Ban, and D.N. Nguyen, "A Meta-Heuristic Algorithm Combining between Tabu and Variable Neighborhood Search for the Minimum Latency Problem", *J. FI*, Vol. 156, No. 1, 2017, pp. 21-41

[4] H.B. Ban, "Using Metaheuristic for Solving the Resource-Constrained Deliveryman Problem", *Proc. SOICT*, 2019, pp. 917-935.

[5] A. Lucena, "Time-dependent Traveling Salesman Problem-the Deliveryman case", *J. Networks*, Vol. 20, No. 6, 1990, pp. 753-763.

[6] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, "The Minimum Latency Problem", *Proc. STOC*, 1994, pp. 163-171.

[7] M. Dorigo, and T. Stutzle, "Ant Colony Optimization", *Bradford Books*, London, 2004.

[8] M. Fischetti, G. Laporte, S. Martello, "The Delivery-man Problem and Cumulative Matroids", *J. Oper Res*, Vol. 41, No. 6, 1993, pp. 1055–1064.

[9] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley, Reading, Massachusetts*, 1989.

[10] H. Hasegawa, "Optimization of GROUP behavior, Japan Ethological Society Newsletter", Vol. 43, 2004, pp. 22-23.

[11] D. Karaboga, “An Idea Based on Honeybee Swarm for Numerica Optimization”, *Technical Report TR06, Erciyes University*, 2005.

[12] J. Kennedy, R. Eberhart, “Particle Swarm Optimization”, *Conf. Neural Networks*, 1995, pp. 1942–1948.

[13] X. Li, J. Liao, and M. Cai, “Ant Colony Algorithm for Large Scale TSP”, *Proc. ICECING*, 2011, pp.573-576.

[14] M. Manfrin, M. Birattari, T. Stützle, and Marco Dorigo, “Parallel Ant Colony Optimization for the Traveling Salesman Problem”, *Proc. ANTS*, 2006, pp 224-234.

[15] I. Mendez-Diaz, P. Zabala, A. Lucena, “A New Formulation for the Traveling Deliveryman Problem", *J. Discret Appl Math*, No. 156, 2008, pp. 3223-3237.

[16] N. Mladenovic, P. Hansen, “Variable Neighborhood Search", *J. Operations Research*, Vol. 24, 1997, pp. 1097-1100.

[17] N. Mladenovic, D. Urosevi, and S. Hanafi, “Variable Neighborhood Search for the Travelling Deliveryman Problem", *J. 4OR*, Vol. 11, 2012, pp. 1-17.

[18] M. Reimann, K. Doerner, and R.F. Hartl, “Analyzing a Unified Ant System for the VRP and Some of Its Variants”. *S. Cagnoni et al. (Eds.): EvoWorkshops, LNCS*, pp. 300-310, 2003.

[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*, Vol. 9, No. 2, 2011, pp. 189-209.

[20] XS. Yang, S. Deb, “Cuckoo Search Via Levy Flights”, *Proc. NaBIC*, 2009, pp. 210–214.

[21] 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.*, 1997, pp. 215-310.

[22] H. X. Huan, N. Linh-Trung, D. Duc-Dong, and T. Huynh, “Solving the Traveling Salesman Problem with Ant Colony Optimization: A Revisit and New Efficient Algorithms”, *J. REV*, Vol. 2, No. 3–4, 2012, pp. 121-129.

[23] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.

Ha-Bang Ban Ha-Bang Ban received his B.E. in Information Technology, and Ph.D in Computer Science at Hanoi University of Science and Technology (HUST), Vietnam in 2006, 2015, respectively. He is currently a lecturer at the School of Information and Communication Technology (SOICT), HUST, Vietnam. Research interests of Dr. Ban include algorithms, graphs, optimization, and logistics, etc. He has published many publications in peer-reviewed international journals and conferences.