Journal of Computer Science and Cybernetics, V.36, N.3 (2020), 233–250
DOI 10.15625/1813-9663/36/3/14770

EFFICIENT METAHEURISTIC ALGORITHMS FOR THE MULTI-STRIPE TRAVELLING SALESMAN PROBLEM

HA-BANG BAN

[1]*School of Information and Communication Technology, Hanoi University of Science and Technology*

Abstract. The Multi-stripe Travelling Salesman Problem (Ms-TSP) is an extension of the Travelling Salesman Problem (TSP). In the q-stripe TSP with $q \geq 1$, the objective function sums the costs for traveling from one vertex to each of the next q vertices along the tour. To solve medium to large-sized instances, a metaheuristic approach is proposed. The proposed method has two main components, which are construction and improvement phases. The construction phase generates an initial solution using the Greedy Randomized Adaptive Search Procedure (GRASP). In contrast, the optimization phase improves it with several variants of Variable Neighborhood Search (VNS), both coupled with a technique called Shaking Technique to escape from local optima. Besides, the Adaptive Memory (AM) technique is applied to balance between diversification and intensification. To show the efficiency of our proposed metaheuristic algorithms, we extensively implement them on benchmark instances. The results indicate that the developed algorithms can produce efficient and effective solutions at a reasonable computation time.

Keywords. q-stripe-TSP; Adaptive memory; VND, VNS; GVNS.

1. INTRODUCTION

The Travelling Salesman Problem (TSP) has been studied in a number of previous works [4]. Given an complete graph $K_n = (V, E)$, where $V = 1, 2, \ldots n$ is a set of vertices, and E is the set of edges. Let $C = \{c_{ij} \,|\, c_{ij} \geq 0\}$ be a symmetric cost matrix in which c_{ij} is the distance between vertex i and j. The goal is to find a solution $T = (\tau(1), \tau(2), ..., \tau(n))$ that minimizes the total travel length $\sum_{i=1}^{n}(c(\tau(i), \tau(i+1))$. The term $\tau(n+1)$ in the above formula coincides with $\tau(1)$. That means the salesman returns to the vertex from which he began his tour. In this paper, a new variant of TSP, that is the q-stripe TSP with $1 \leq q \leq (n-1)/2$ is studied. This problem was already introduced by E. Cela et al. in [4]. In this variant, the goal is to find a permutation $T = (\tau(1), \tau(2), ..., \tau(n))$ over all vertices that minimizes the cost function as follows

$$L(T) = \sum_{p=1}^{q}\sum_{i=1}^{n}(c(\tau(i), \tau(i+p)),$$

where the term $\tau(i+p)$ for $i \geq n$ coincides $\tau(i+p-n)$. As for $q = 1$, the problem becomes the classical TSP. In the other cases, the permutation T encodes a tour through the vertices, and the objective function sums the distances c_{ij} over all vertices i and j that are at most q steps away from each other when travelling along the tour.

*Corresponding author.
E-mail address: BangBH@soict.hust.edu.vn

In [4], E. Cela et al. also indicate that the q-stripe-TSP is a particular case of the Quadratic Assignment Problem (QAP) [7], that has many practical applications such as in transportation [1], distributed computing, balancing of turbine running [10], reactionary chemistry [15], genetics [16], creating the control panels, manufacture [9], scheduling problem [8]. Assume that, on a new campus the buildings are arranged to minimize the total walking distances for students. The model can be described as follows: Let n be the number of the new buildings to be established, and let c_{ij} be the distance between location i and location j on the campus, where the new buildings can be established. The connection matrix $f = (f_{ij})$ describes the frequencies with which students walk between locations i and j. An assignment of the sites to the buildings is wanted, i.e. a permutation τ of the set $\{1, 2, ..., n\}$. The product $f_{ij} \times c_{\tau(i)\tau(j)}$ describes the weighted walking distance between buildings $\tau(i)$ and $\tau(j)$ if building $\tau(i)$ is established on location i and building $\tau(j)$ is established on location j. Therefore, the problem to minimize the total walking distance becomes $\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \times c_{\tau(i)\tau(j)}$. The problem has been known as the Quadratic Assignment Problem (QAP). E. Cela et al. [4] show that the q-stripe TSP problem is a special case of the QAP on the graph C_n^q (note that: The graph C_n^q generated from the graph G easily [4], encodes the cost structure of the q-stripe TSP on n vertices). By making matrix C the distance matrix of n vertices and by making matrix f_{ij} the adjacency matrix of the graph C_n^q, we arrive at the q-stripe TSP as a particular case of the QAP.

For example: Given a complete graph $K_6 = \{1, 2, 3, 4, 5, 6\}$ that consists of 6 locations from L_1 to L_6, and 6 buildings. In the case of $q = 2$, the graph C_6^2 is generated from K_6 according to [4] as follows: For $q \geq 1$ and $n \geq 2q + 1$, the vertex set of graph C_6^2 is $\{1, 2, 3, 4, 5, 6\}$, and there is an edge between any two distinct vertices i and j with $\mid i-j \mid \leq q$ or $\mid i - j \mid \geq n - q$ (see in Figure 1). In Figure 1, K_6 includes all edges while C_6^2 does not consists of the blocked edges (the blocked edges are illustrated in dashed lines). The blocked edge (L_2, L_5) means that there is no any direct connection between buildings if they are established on L_2, and L_5, respectively. It is suitable in practical situation when there is no way for students to walk directly from one building to another. The q-stripe TSP becomes the QAP on the graph C_6^2.

Assume that, $T = \{3, 2, 4, 1, 5, 6\}$ is a solution and its corresponding objective value

$$QAP(T) = c_{32} + c_{34} + c_{35} + c_{36} + c_{24} + c_{21} + c_{26} + c_{41} + c_{45} + c_{15} + c_{16} + c_{56}.$$

$$L(T) = c_{32} + c_{24} + c_{41} + c_{15} + c_{56} + c_{63} + c_{34} + c_{21} + c_{45} + c_{16} + c_{53} + c_{62}.$$

Obviously, the objective value of the q-stripe-TSP is equal to the one of the QAP because C is the symmetric matrix. This implies that the q-stripe TSP problem is a special case of the QAP on the graph C_n^q.

The q-stripe-TSP is also NP-hard because it is a generalization case of the TSP. For NP-hard problems, there are three common approaches to solve them, namely, 1) exact, 2) α-approximation algorithm, 3) heuristic (or metaheuristic). Firstly, the exact algorithms guarantee to find the optimal solution and take exponential time in the worst case, but they often run much faster in practice. Several exact algorithms that belong to this approach are Branch-and-Bound, Branch-and-Cut, Branch-and-Price, and Dynamic Programming [17]. For the q-stripe-TSP, there exists no work in the literature to solve the problem. However, numerous exact algorithms solve the TSP with large sizes [17]. Secondly, the term "α-approximation algorithm" refers to algorithms that produce a solution within some factor

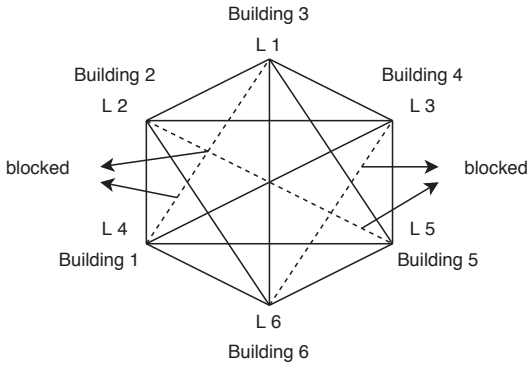

Figure 1. The graph K_n and C_n^q

of α of the optimal solution. Currently, there is no approximation algorithm proposed for the problem. In the case of the TSP, the best-known approximation ratio of $\frac{3}{2}$ can be found in [4]. However, the ratio is still large for practical applications. Finally, (meta)-heuristic algorithms perform well in practice, and the efficiency of them can be evaluated by experiments. Heuristics are often too greedy; therefore, they usually get trapped in a local optimum and thus fail to obtain the global optimum solution. The metaheuristic approach, on the other hand, is not greedy and may even accept a temporary deterioration of solution, which allows them to explore more thoroughly the solution space and thus to get better solutions.

Previously, research on the q-stripe-TSP has been not studied much, and this work presents the first metaheuristic approach for this problem. Our metaheuristic algorithm is mainly based on the principles of systematically exploring several different neighborhoods in the VNS [14], and GRASP [6] to solve the problem. In a construction phase, the GRASP is used to build an initial solution that is the input for an improvement phase. In a cooperative way, several variants of VNS [14] combined with Shaking techniques [12] are employed to generate various neighborhoods as well as allow the search to escape from local optimal in the improvement phase. In addition, Adaptive Memory (AM) [13] is integrated into our algorithms to balance between diversification and intensification. Extensive numerical experiments on benchmark instances show that the proposed algorithm reaches the efficient and effective solutions at a reasonable amount of time.

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

2. METHODOLOGY

The efficient metaheuristic includes the GRASP [6] in the construction phase, and several variants of VNS (VND, VNS, and GVNS) [14], Shaking technique [12], as well as Adaptive Memory (AM) [13] in the improvement phase, respectively. The good metaheuristic algorithm needs to keep the balance between intensification and diversification. Diversification means to create various solutions to explore the solution space on a global scale. In contrast,

intensification means to focus on the search in a local region by exploiting the information that a current good solution is found in this region. In the algorithm, several variants of VNS ensure intensification while the Shaking and AM techniques keep diversification. This combination maintains the simplicity spirit of several variants of VNS, while it explores the solution space effectively.

- The GRASP is introduced by Feo et al. [6]. The basic idea of GRASP allows us to balance between greedy and random approaches. The advantage of the GRASP compared to other heuristics, such as Ant Colony Algorithm, Genetic Algorithm,... in [3], is that there is the only parameter to tune (the size of the candidate list). The GRASP appears to be competitive with respect to the quality of solutions, and the fact that it is easier to implement and tune.

- The Variable Neighborhood Search (VNS) algorithm is proposed by Mladenovic et al. [14]. It executes alternately local search procedure, and shaking technique to escape from the local optima. At each iteration, a random solution is generated from a current neighborhood, and then a local search procedure is implied to improve the solution. If the new solution is better than the best one, the procedure is repeated. Otherwise, the search passes to the next neighborhood.

- The Variable Neighborhood Descent (VND) algorithm, which is a VNS variant, is proposed by Mladenovic et al. [14]. The VND is obtained if a change of neighborhoods is performed in a deterministic way. Assume that, an initial solution is given. Local search heuristics in their descent phase is used to generate neighborhoods. The final solution should be a local minimum with respect to all neighborhoods. The difference between the VNS and VND is that the VNS uses Shaking.

- The General Variable Neighborhood Search (GVNS) algorithm [14] is a variant of VNS. It includes an initial feasible solution, and a shaking procedure followed by VND local search. The GVNS is a VNS variant where the VND is used as the improvement procedure.

- The Adaptive Memory (AM) is a technique used in the local search [13]. The technique allows first to diversify the search by exploring solutions that are very different from each other, second to intensify the search to identify better local optima in a promising region. However, since the technique does not maintain enough diversification, the shaking technique is used. It allows guiding the search towards an unexplored part of the solution space. The combination helps to balance between diversification and intensification.

An outline of the GRASP, VND, VNS, GVNS, and GVNS with AM (GVNS-AM) are shown in Algorithms from 1 to 5. These algorithms fall into a single-start version. Moreover, we develop a multi-start version for them (see Algorithm 6). It simply executes the variants of VNS for the number of iterations and returns the best-found solution.

2.1. The construction phase

Algorithm 1 shows the constructive procedure. The algorithm implements iteratively until an initial solution is found. At each step, a Restricted Candidate List (RCL) is deter-

Algorithm 1 Construction

Input: v_1, V, α are a depot (or starting vertex), vertex set, and the size of RCL, respectively.
Output: The initial solution T.

1: $\{v_1$ is a depot$\}$
2: $T \leftarrow \{v_1\}$;
3: **repeat**
4: $\{v_c$ is the last vertex in $T\}$
5: Generate RCL which consists of α nearest vertices to v_c;
6: Pick a random vertex $v \in \{v_i | v_i \in RCL$ and $v_i \notin T\}$;
7: $T \leftarrow \{v_i\}$;
8: **until** $|T| < n$;
9: return T;

Algorithm 2 VND

Input: T, k_m are an initial solution, and neighborhood set, respectively.
Output: T^* $\{T^*$ is the best solution$\}$

1: $k = 1$;
2: **repeat**
3: Find the best neighborhood $T^{'}$ of $T \in N_k(T)$; {implement local search}
4: **if** $(L(T^{'}) < L(T))$ or $(L(T^{'}) < L(T^*))$ **then**
5: $T = T^{'}$;
6: **if** $(L(T^*) > L(T^{''}))$ **then**
7: $T^* = T^{'}$;
8: **end if**
9: $k \leftarrow 1$; {return to the first neighborhood}
10: **else**
11: $k++$; {switch to the next neighborhood}
12: **end if**
13: **until** $k < k_m$;
14: $T^* = T^{'}$;
15: **return** T^*;

mined by ordering all non-selected vertices in terms of a greedy function that measures the benefit of including them in the tour. After that, one element will be chosen randomly from RCL to add to the partial solution. Since all vertices are visited, the algorithm stops, and the initial solution is returned. The size of RCL is a parameter that controls the balance between greediness and randomness.

2.2. Neighborhoods

We use seven neighborhoods widely proposed in the literature to explore the search space of the problem [10]. Let N_k $(k = 1, ..., k_m)$ be a finite set of pre-selected neighborhood structures, and let $N_k(T)$ be the set of solutions in the $k-th$ neighborhood of T. We describe in more detail about seven neighborhoods:

1) **move-up** (N_1) moves a vertex forward one position in T. The complexity of exploring

Algorithm 3 VNS

Input: T, k_m are an initial solution, and neighborhood set, respectively.
Output: T^*. {T^* is the best solution}

$k = 1$;
repeat
  $T^{'}$ = Shaking-Technique(T);
  Find the best neighborhood $T^{''}$ of $T \in N_k(T^{'})$; {local search}
  if $(L(T^{'}) < L(T))$ or $(L(T^{'}) < L(T^*))$ **then**
    $T = T^{''}$;
    if $(L(T^*) > L(T^{''}))$ **then**
      $T^* = T^{'}$;
    end if
    $k \leftarrow 1$; {return to the first neighborhood}
  else
    $k++$; {switch to the next neighborhood}
  end if
until $k < k_m$;
$T^* = T^{'}$;
return T^*;

Algorithm 4 GVNS

Input: T, k_m, t_{max} are a starting solution, neighborhood set, and the maximum running time, respectively.
Output: T^*. {T^* is the best solution}

repeat
  $k = 1$;
  while $(k < k_m)$ **do**
    $T^{'}$ = Shaking-Technique(T);
    {deploy VND procedure}
    $T^{''} \leftarrow$ VND($T^{'}, k_m$);
    if $(L(T^{''}) < L(T))$ or $(L(T^{''}) < L(T^*))$ **then**
      $T = T^{''}$;
      if $(L(T^*) > L(T^{''}))$ **then**
        $T^* = T^{''}$;
      end if
      $k \leftarrow 1$; {return to the first neighborhood}
    else
      $k++$; {switch to the next neighborhood}
    end if
  end while
until $time < t_{max}$
$T^* = T^{'}$;
return T^*;

Algorithm 5 GVNS with AM

Input: T, k_m, t_{max} are a starting solution, neighborhood set, and the maximum running time, respectively.

Output: T^*. {T^* is the best solution}

repeat
  $k = 1$;
  while $(k < k_m)$ **do**
    $T^{'} =$ Shaking-Technique(T);
    {deploy VND procedure}
    $T^{''} \leftarrow$ VND($T^{'}, k_m$);
    if $(L(T^{''}) < L(T))$ or $(L(T^{''}) < L(T^*))$ **then**
      $T = T^{''}$;
      if $(L(T^*) > L(T^{''}))$ **then**
        $T^* = T^{''}$;
      end if
      $k \leftarrow 1$; {return to the first neighborhood}
    else
      $k++$; {switch to the next neighborhood}
    end if
    $AM \leftarrow \{T^{''}\}$;
    if $(\mid AM \mid == m)$ **then**
      Erase AM;
    end if
    $T =$ Pick the best tour in AM in accordance with (1);
  end while
until $time < t_{max}$
$T^* = T^{'}$;
return T^*;

Algorithm 6 multi-start version

Input: $v_1, V, \alpha, k_m, t_{max}, m\_start\_iter$ are a starting vertex, vertex set, size of RCL, neighborhood set, maximum running time, and number of starts, respectively.

Output: the best solution T^*.

$i = 0$;
repeat
  $T =$ Construction(v_1, V, α);
  $T^{'} =$ GVNS-AM(T, k_m, t_{max});{The same for the VND, VNS, GVNS}
  if $(L(T^*) > L(T^{'}))$ **then**
    $T^* = T^{'}$;
  end if
  $i++$;
until $i \leq m\_start\_iter$
return T^*;

Algorithm 7 Shaking-Technique(T)

Input: T is the tour.
Output: a new tour T.

$k_1 = 1 + \text{rand}(\frac{n}{4})$;
$k_2 = k_1 + 1 + \text{rand}(\frac{n}{4})$;
$k_3 = k_2 + 1 + \text{rand}(\frac{n}{4})$;
$\{T_1$ copies consecutive vertices from 1-st to $k_1 - th$ position in $T\}$
$T_1 = T[1 : k_1]$;
$\{T_2$ copies consecutive vertices from $k_3 - th$ to $k_4 - th$ position in $T\}$
$T_2 = T[k_3 : k_4]$;
$\{T_3$ copies consecutive vertices from $k_2 - th$ to $k_3 - th$ position in $T\}$
$T_3 = T[k_2 : k_3]$;
$\{T_4$ copies consecutive vertices from $k_1 - th$ to $k_2 - th$ position in $T\}$
$T_4 = T[k_1 : k_2]$;
$T = T_1 \cup T_2 \cup T_3 \cup T_4$;
return T;

the neighborhood is $O(Tsol \times n)$;

2) **move-down** (N_2) moves a vertex backward one position in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$;

3) **shift** (N_3) relocates a vertex to another position in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$;

4) **swap-adjacent** (N_4) attempts to swap each pair of adjacent vertices in the tour. The complexity of exploring the neighborhood is $O(Tsol \times n)$;

5) **swap** (N_5) tries to swap the positions of each pair of vertices in the tour. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$;

6) **2-opt** (N_6) removes each pair of edges from the tour and reconnects them. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$;

7) **Or-opt** (N_7) is reallocated three adjacent vertices to another position of the tour. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$.

2.3. Local search

To improve the solution, we developed local search procedure by combining the seven neighborhood structures. Assume that, an initial solution is given. Local search heuristics are used to generate neighborhoods. The final solution should be a local minimum with respect to all neighborhoods. The order of neighborhoods is fixed. In a pilot study, we found that the performance of the algorithm is relatively insensitive to the order in which the neighborhoods are used. The neighborhoods are therefore explored in a specific order, from "small" to "large" as it is common, i.e., swap-adjacent, move-up, move-down, remove-insert, swap, 2-opt, and or-opt.

2.4. Shaking technique

The shaking mechanism design is very important to achieve success in our algorithm. If the mechanism produces too small perturbation moves, the search procedure may return to the previously visited local optimum points. On the other hand, excessive shaking moves may drive the search procedure to undesirable regions in the search space. In this article, we implement an adaptive perturbation mechanism. The shaking mechanism, called double-bridge, was originally developed in [12]. It consists of removing and re-inserting four arcs in such a way that a feasible tour is generated. This mechanism can also be seen as a permutation of two disjoint segments of a tour. The detail is described in Algorithm 7.

2.5. Adaptive memory technique

The Adaptive Memory Technique (AM) [13] is a dynamic memory that changes at each iteration. It saves various solutions obtained by the local search step. For each solution in the AM, we count its cost and diversity in a set of solutions in the AM:

$$R(T) = (|AM| - RF(T) + 1) + \beta \times (|AM| - RD(T) + 1), \tag{1}$$

where $|AM|$ is the current size of AM; $\beta \in [0, 1]$; $RF(T)$ is the rank of T according to its objective function; $RD(T)$ is the rank of T according to its diversity value,

$$\overline{d}(T) = \frac{\sum_{k=1}^{n} d(T, T_k)}{n}, \tag{2}$$

where $d(T, T_k)$ is the metric distance between T and T_k, and $\overline{d}(T)$ is the average metric distance of T in the AM list. In intuitive way, the distance is the minimum number of transformations from T to T_k. When there exists no polynomial operator for calculating $d(T, T_k)$, $d(T, T_k)$ is n minus the number of vertices which has the same position in both T, and T_k. The larger $\overline{d}(T)$ is, the higher $RD(T)$ is. The smaller $L(T)$ is, the higher $RF(T)$ is. The solution that has the largest $R(T)$ value is selected from AM.

2.6. Stop condition

The VND, and VNS stop if after all neighborhoods are implemented, the improvement cannot be found while the GVNS, and GVNS-AM stop after t_{max} seconds or the best-solution is found (t_{max} is the parameter of the algorithm and its value is determined from preliminary experiments).

3. EVALUATIONS

Our algorithms are run on a Pentium 4 core i7 2.40 GHz processor with 8 GB of RAM. After preliminary experiments, the parameters RCL, $\beta, |AM|$, t_{max}, and m_start_iter are respectively set to 10, 0.75, 100, 10000, and 5. These parameters are chosen through empirical tests, and with them, the algorithm seems to produce good solutions at a reasonable amount of time in comparison with other parameter values tested. We also compare the multi-start version of our algorithm with single-start, in which ten solutions are generated with the GRASP, each of which is improved by using the VND, VNS, GVNS, and GVNS-AM. The

best solutions found by ten independent runs are reported. In the tables below, we give a report on the time when the best solution of each instance is reached.

To evaluate our algorithm's solution quality, we need to compare it with the other metaheuristics. The main problem is that there exists no other metaheuristic reported in the literature for this problem. That means we found no previous attempts to solve the q-stripe-TSP, neither exact nor heuristic (or metaheuristic), to compare with. Therefore, to evaluate the efficiency of our algorithm, 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 upper bound (UB) obtained by the Nearest Neighborhood Search in [15]. The Nearest Neighborhood Search is not promising theoretically; however, it yields good enough solution in practice. In addition, to demonstrate the efficiency and wide applicability of the algorithms in the case of TSP, our solutions are also compared to the optimal solutions in some TSP-instances though it is not designed to solve the TSP. The optimal solutions for the TSP-instances are extracted in [18].

3.1. Instances

We implement the algorithm in random and real datasets as follows:

- To create a set of 80 instances, we have generated 20 random instances for each problem size ($n = 30, 50, 100$, and 200). For each instance, vertex coordinates have been generated from a uniform distribution between 0 and 100. All distances are Euclidean, rounded down to the nearest integer. The instances are available from the author upon request.

- The real datasets are extracted from TSPLIB in [18].

3.2. Results

We define the improvement of our algorithm with respect to BKS (BKS is the best solution found by our algorithm) in comparison with the solution from the Nearest Neighborhood Search (UB) [10] for the problem as followings

$$Gap[\%] = \frac{BKS - UB}{UB} \times 100\%.$$

In tables, *BKS*, *Time* correspond to the best solution, and average time in seconds of ten executions obtained by all algorithms, respectively. The values in Table 1 are the average values from Table 5 to 8 in Appendix while the values in Table 2 are the average values from Table96 to 12 in Appendix. In Table 3, the OPT column corresponds to the optimal TSP-solutions using the q-stripe-TSP's cost function. Table 4 shows the results of our algorithms in comparison with the optimal TSP-solutions for some instances. In Table 4, *Gap* is calculated as follows

$$Gap[\%] = \frac{BKS - OPT}{OPT} \times 100\% \quad (OPT \text{ is the optimal value}).$$

From Tables 1 and 2, it can also be seen that the difference in the average gap between

Table 1. Average results for random instance in single-start

n	VND		VNS		GVNS		GVNS-AM	
	Gap [%]	*Time*	*Gap* [%]	*Time*	*Gap* [%]	*Time*	*Gap* [%]	*Time*
30	-10.00	0.026	-11.79	0.034	-11.82	0.065	-11.85	0.077
50	-10.76	1.27	-12.47	1.30	-12.86	2.24	-12.91	2.24
100	-11.08	9.19	-11.40	10.79	-11.92	18.84	-12.32	19.79
200	-11.27	367.17	-12.26	367.25	-12.31	377.07	-12.65	377.82

Table 2. Average results for random instance in multi-start

n	VND		VNS		GVNS		GVNS-AM	
	Gap [%]	*Time*	*Gap* [%]	*Time*	*Gap* [%]	*Time*	*Gap* [%]	*Time*
30	-10.00	0.036	-11.79	0.052	-11.82	0.084	-11.85	0.114
50	-10.76	3.23	-12.47	4.23	-12.86	5.23	-12.91	5.23
100	-11.08	105.83	-11.40	127.78	-11.92	277.38	-12.32	227.79
200	-11.27	3671.73	-12.26	3720.53	-12.31	4171.46	-12.65	4177.34

Table 3. Comparison of our best-found q-stripe-TSP-solution with the optimal TSP-solution using the q-stripe-TSP's objective function

Instances	Optimal TSP using the q-stripe-TSP's objective function	Our best-found q-stripe-TSP	Difference [%]
st70	123865	109892	11.28
berlin52	762862	641005	15.97
eil51	42073	36774	12.59
eil76	95211	82474	13.38
pr76	19477018	18400099	5.53
KroA100	8840395	7232665	18.19
eil101	173251	149660	13.62
ch130	3044215	2603447	14.48
Aver			13.13

Table 4. Comparison with the optimal solutions for TSP-instances

Instances	*OPT*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
st70	675	701	3.85	7.56	687	1.78	8.54	678	0.44	9.54	**675**	0.00	9.14
eil51	426	452	6.10	1.35	428	0.47	1.4	428	0.47	2.34	428	0.47	2.43
eil76	538	567	5.39	8.12	555	3.16	7.25	555	3.16	8.15	553	2.79	8.21
berlin52	7542	7970	5.67	1.4	7606	0.85	1.41	7542	0.00	1.47	**7542**	0.00	1.52
kroA100	21282	22586	6.13	9.23	21735	2.13	11.45	21735	2.13	20.12	21308	0.12	19.21
pr107	44303	46707	5.43	12.12	46707	5.43	14.51	45146	1.90	18.21	45002	1.58	17.65
ch130	6110	6442	5.43	112.32	6341	3.78	124.65	6341	3.78	165.23	6275	2.70	162.32
pr76	108159	110792	2.43	7.89	109219	0.98	7.35	109219	0.98	8.45	109094	0.86	8.54
gr17	2085	**2085**	0.00	0	**2085**	0.00	0	**2085**	0.00	0	**2085**	0.00	0
gr21	2707	**2707**	0.00	0	**2707**	0.00	0	**2707**	0.00	0	**2707**	0.00	0
gr24	1272	**1272**	0.00	0	**1272**	0.00	0	**1272**	0.00	0	**1272**	0.00	0
gr48	5046	5046	0.00	1.21	5046	0.00	1.32	5046	0.00	2.25	**5046**	0.00	2.25
Aver			3.37	13.43		1.55	14.82		1.07	19.65		0.71	19.27

the construction phase and improvement phase for the VND, VNS, GVNS, and GVNS-AM is from 10.0% to 12.91% for both of single-start and multi-start. This indicates that the construction phase returns good quality solutions fast. Although the improvement of the post phase upon the construction one is not too large, it is significant. The VNS, GVNS, and GVNS-AM obtain better solutions than the VND from 0.99% to 1.38%. It is easy to understand when the VND only implements intensification. On the other hand, the others maintain diversification better by using the shaking technique. Therefore, the shaking technique plays an important role in improving the quality of solution. The solutions obtained by the GVNS are usually slightly better than those of the VNS, but the running time of the GVNS consumes more than VNS. It implies that we cannot improve further the quality of solution only by running the algorithm with a larger amount of time.

In Tables 1 and 2, we can draw some conclusions about the working of our algorithm. Unsurprisingly, the multi-start versions require a much larger computation time than the single-start versions, but the quality improvement is not found. This may indicate that the GRASP in the multi-start versions is not able to provide enough diversification and the shaking is useful.

In all tables, the GVNS-AM obtains better solutions than the GVNS does. Clearly, the AM is an efficient technique when it ensures the balance between intensification and diversification. The technique allows first to diversify the search by exploring solutions that are very different from each other, and second to intensify the search to identify better local optima in a promising area.

Table 3 shows unsurprisingly that the optimal TSP-solutions are generally not good solutions to the q-stripe-TSP in the same instances. On average, the best solutions found by our algorithm are about 13.13% better than the optimal TSP solutions by using the q-stripe-TSP's objective function. Therefore, the methods designed for the TSP may not be adapted easily to solve the q-stripe-TSP. Developing the efficient algorithms for the q-stripe-TSP is necessary.

Table 4 shows that our algorithm can run well to the TSP (note that the TSP is a particular variant of the q-stripe-TSP since q=1) although it is not designed to solve it. In comparison with the optimal solutions in the TSP, our algorithm's solutions are the optimal solutions for the instances with up to 70 vertices. The average gap between the optimal solution and our result is about 0.71% for the instances with 130 vertices. It shows that our results are near to the optimal values.

4. DISCUSSIONS

Three types of algorithms are used to solve NP-hard problem. The first type consists of exact algorithms, but they are very time-consuming for large instances. The second type consists of $\alpha-$approximation algorithms that produce a solution within the factor of α of the optimal solution. The third type includes heuristic (or metaheuristic) algorithms. These algorithms can provide good solutions within a short computation time for large instances. In addition, they are easy and fast to implement.

Currently, numerous works to solve the TSP are proposed. We immediately think that a good algorithm for the TSP can solve the q-stripe-TSP well by using the q-stripe-TSP's objective function. However, the experimental results in Table 3 show that the methods designed for the TSP may not be adapted easily to solve the q-stripe-TSP. Therefore, developing efficient algorithms for the q-stripe-TSP is necessary. We found no previous works in the literature to solve the q-stripe-TSP, neither exact nor heuristic. Our contribution is to provide the efficient algorithms. These algorithms are the first metaheuristics for the problem. In the work, four algorithms include the VND, VNS, GVNS, and GVNS-AM. These algorithms often solve the optimization problem effectively. We can find the similar approach in [1]. Among the algorithms, the VND gives the worse results. It is easy to understand because the VND only ensures intensification while the others keep diversification by using the shaking technique. The experimental results also show that it is impossible to improve the quality of solution by running the algorithm for a long time. More specifically, the GVNS returns very slightly better solutions than those of the VNS. The GVNS-AM outperforms than the others. Obviously, the AM and Shaking technique bring the efficiency well since it balances between diversity and intensification. It maintains the simplicity spirit of the GVNS while it explores the solution space effectively. The multi-start versions do not provide better solutions than single-start in many cases while they consume much time. This may indicate that the GRASP in the multi-start version is not able to provide enough diversification.

The VND, VNS, GVNS, and GVNS-AM seem to work well. We divide them into two types of algorithms: 1) Fast algorithms include the VND, and VNS. They use significantly less computing time, combined with a rather small loss of solution quality. Both algorithms are useful for the large instances; 2) Slow algorithms consist of the GNVS, and GVNS-AM. They are the most effective algorithms in terms of solution quality for the q-stripe-TSP as well as TSP, although they are quite a time consuming for the problem with up to 200 vertices. In the TSP-instances, they can find the optimal solutions for the instances with up to 70 vertices. Moreover, it also provides the near-optimal solutions for the larger problems in Table 4. Although our purpose is not to provide metaheuristics for the TSP, the obtained results for this problem show the efficiency and wide applicability of our algorithms.

5. CONCLUSIONS

In this work, we provide several metaheuristics for the q-stripe-TSP. These metaheuristics consist of the GRASP in construction phase and the VND, VNS, GVNS, and GVNS-AM in improvement phase. Besides, the AM technique is applied to balance between diversification and intensification. Experiments show that the metaheuristics produce good solutions for the problem at a reasonable amount of time. For the case of the TSP (a particular case of the q-stripe-TSP), the optimal solutions can be obtained for the instances with 70 vertices. In the future, we intend to extend our algorithm by including more neighborhoods and carefully studying the effectiveness of each neighborhood on the q-stripe-TSP. Increasing the efficiency of our algorithm, even more, to allow even larger problems to be solved, is another future research topic.

APPENDIX

Table 5. Average results for q-TSP-30-x instance in single-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-30-1	22397	19949	-10.93	0.03	19402	-13.37	0.04	19402	-13.37	0.06	19402	-13.37	0.08
TSP-30-2	22370	20163	-9.87	0.01	19812	-11.43	0.04	19811	-11.44	0.07	19811	-11.44	0.08
TSP-30-3	20412	17942	-12.10	0.02	17745	-13.07	0.04	17745	-13.07	0.06	17745	-13.07	0.08
TSP-30-4	23811	21245	-10.78	0.03	21093	-11.41	0.03	21078	-11.48	0.07	21067	-11.52	0.08
TSP-30-5	21664	19363	-10.62	0.04	19136	-11.67	0.03	19094	-11.86	0.06	19094	-11.86	0.08
TSP-30-6	21021	19235	-8.50	0.03	18940	-9.90	0.06	18940	-9.90	0.06	18938	-9.91	0.07
TSP-30-7	23784	21535	-9.46	0.03	21081	-11.36	0.04	21081	-11.36	0.07	21081	-11.36	0.08
TSP-30-8	23033	20631	-10.43	0.03	20024	-13.06	0.03	20024	-13.06	0.07	20024	-13.06	0.08
TSP-30-9	21331	19458	-8.78	0.02	19050	-10.69	0.03	19050	-10.69	0.07	19050	-10.69	0.07
TSP-30-10	21867	19425	-11.17	0.01	19167	-12.35	0.03	19167	-12.35	0.08	19167	-12.35	0.08
TSP-30-11	20048	18301	-8.71	0.05	17645	-11.99	0.02	17645	-11.99	0.07	17643	-12.00	0.07
TSP-30-12	18507	16921	-8.57	0.03	16638	-10.10	0.02	16638	-10.10	0.06	16581	-10.41	0.08
TSP-30-13	22476	20547	-8.58	0.02	19719	-12.27	0.02	19743	-12.16	0.06	19718	-12.27	0.08
TSP-30-14	21060	19495	-7.43	0.01	18833	-10.57	0.03	18833	-10.57	0.06	18833	-10.57	0.07
TSP-30-15	22085	20091	-9.03	0.02	19581	-11.34	0.04	19578	-11.35	0.06	19577	-11.36	0.07
TSP-30-16	20826	18892	-9.29	0.02	18663	-10.39	0.04	18660	-10.40	0.07	18660	-10.40	0.08
TSP-30-17	22753	19748	-13.21	0.04	19388	-14.79	0.03	19388	-14.79	0.06	19388	-14.79	0.08
TSP-30-18	20686	18788	-9.18	0.02	18505	-10.54	0.03	18505	-10.54	0.07	18505	-10.54	0.08
TSP-30-19	22352	19194	-14.13	0.03	19135	-14.39	0.04	19045	-14.80	0.06	19011	-14.95	0.07
TSP-30-20	21647	19657	-9.19	0.03	19241	-11.11	0.04	19241	-11.11	0.06	19241	-11.11	0.08
Aver			-10.00	0.026		-11.79	0.034		-11.82	0.065		-11.85	0.077

Table 6. Average results for q-TSP-50-x instance in single-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-50-1	65007	59789	-8.03	1.26	58992	-9.25	1.30	58543	-9.94	2.14	58543	-9.94	2.13
TSP-50-2	66861	60703	-9.21	1.29	59301	-11.31	1.28	58764	-12.11	2.18	58764	-12.11	2.32
TSP-50-3	63723	57928	-9.09	1.27	56460	-11.40	1.31	56507	-11.32	2.30	56460	-11.40	2.33
TSP-50-4	69108	62074	-10.18	1.28	61014	-11.71	1.28	61063	-11.64	2.13	61014	-11.71	2.30
TSP-50-5	64880	60865	-6.19	1.26	59726	-7.94	1.30	59795	-7.84	2.33	59726	-7.94	2.14
TSP-50-6	68108	61450	-9.78	1.28	59466	-12.69	1.28	59466	-12.69	2.29	59466	-12.69	2.18
TSP-50-7	70956	61348	-13.54	1.26	58348	-17.77	1.29	58356	-17.76	2.23	58348	-17.77	2.20
TSP-50-8	68834	63088	-8.35	1.28	61207	-11.08	1.30	61305	-10.94	2.25	61207	-11.08	2.28
TSP-50-9	69326	62014	-10.55	1.28	60338	-12.96	1.31	60517	-12.71	2.17	60338	-12.96	2.15
TSP-50-10	67715	57914	-14.47	1.29	57423	-15.20	1.27	57374	-15.27	2.23	57374	-15.27	2.29
TSP-50-11	63051	53652	-14.91	1.27	53306	-15.46	1.32	52860	-16.16	2.34	52860	-16.16	2.14
TSP-50-12	61257	54530	-10.98	1.25	54234	-11.46	1.31	52922	-13.61	2.25	52922	-13.61	2.27
TSP-50-13	66835	59674	-10.71	1.26	59383	-11.15	1.29	58980	-11.75	2.24	58980	-11.75	2.23
TSP-50-14	67593	60759	-10.11	1.30	58898	-12.86	1.29	58634	-13.25	2.17	58634	-13.25	2.30
TSP-50-15	67333	58442	-13.20	1.26	57098	-15.20	1.29	56635	-15.89	2.23	56635	-15.89	2.28
TSP-50-16	70420	59402	-15.65	1.29	58449	-17.00	1.29	57984	-17.66	2.26	57984	-17.66	2.33
TSP-50-17	68602	60509	-11.80	1.28	60167	-12.30	1.30	59820	-12.80	2.28	59820	-12.80	2.32
TSP-50-18	64936	60462	-6.89	1.30	60297	-7.14	1.30	59822	-7.88	2.21	59822	-7.88	2.20
TSP-50-19	66249	60651	-8.45	1.25	58497	-11.70	1.31	58666	-11.45	2.20	58497	-11.70	2.28
TSP-50-20	67387	58519	-13.16	1.27	58102	-13.78	1.31	57596	-14.53	2.35	57596	-14.53	2.17
Aver			-10.76	1.27		-12.47	1.30		-12.86	2.24		-12.91	2.24

Table 7. Average results for q-TSP-100-x instance in single-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-100-1	254367	229313	-9.85	9.15	229926	-9.61	10.92	227960	-10.38	18.43	227060	-10.74	19.46
TSP-100-2	273942	247520	-9.65	9.25	246762	-9.92	10.71	245285	-10.46	19.00	244086	-10.90	19.95
TSP-100-3	264618	235326	-11.07	9.14	236153	-10.76	11.05	234018	-11.56	18.81	232516	-12.13	19.44
TSP-100-4	264176	232994	-11.80	9.23	232086	-12.15	10.83	230229	-12.85	18.79	229275	-13.21	19.47
TSP-100-5	269213	235668	-12.46	9.23	235220	-12.63	10.69	234974	-12.72	19.12	233222	-13.37	19.82
TSP-100-6	259488	232003	-10.59	9.24	233432	-10.04	11.15	229941	-11.39	18.89	229402	-11.59	19.49
TSP-100-7	268516	238569	-11.15	9.15	235907	-12.14	11.10	236076	-12.08	18.90	233914	-12.89	20.06
TSP-100-8	266177	232567	-12.63	9.18	232273	-12.74	10.84	233214	-12.38	19.09	230559	-13.38	20.06
TSP-100-9	266762	242423	-9.12	9.17	238839	-10.47	10.90	236776	-11.24	19.05	236776	-11.24	19.98
TSP-100-10	267537	233891	-12.58	9.23	233124	-12.86	10.87	231577	-13.44	18.87	231288	-13.55	19.53
TSP-100-11	260790	237922	-8.77	9.19	235817	-9.58	10.57	234600	-10.04	18.55	234044	-10.26	19.93
TSP-100-12	259433	231504	-10.77	9.24	230068	-11.32	10.65	229449	-11.56	18.60	229276	-11.62	19.82
TSP-100-13	252283	222016	-12.00	9.16	220506	-12.60	10.78	217559	-13.76	19.11	217503	-13.79	20.18
TSP-100-14	284030	242532	-14.61	9.17	242509	-14.62	10.59	240937	-15.17	18.43	240305	-15.39	19.92
TSP-100-15	247662	219652	-11.31	9.16	218820	-11.65	11.08	221707	-10.48	18.80	217038	-12.37	20.04
TSP-100-16	279004	249739	-10.49	9.15	248109	-11.07	10.56	244459	-12.38	18.54	244459	-12.38	19.77
TSP-100-17	264287	231400	-12.44	9.24	230987	-12.60	10.59	229537	-13.15	19.18	229190	-13.28	19.75
TSP-100-18	251266	223684	-10.98	9.20	223126	-11.20	10.54	219862	-12.50	18.97	219862	-12.50	20.06
TSP-100-19	262380	236906	-9.71	9.20	236590	-9.83	10.59	234975	-10.44	18.81	233915	-10.85	19.48
TSP-100-20	256873	232339	-9.55	9.16	230850	-10.13	10.75	230233	-10.37	18.78	228810	-10.92	19.52
Aver			-11.08	9.19		-11.40	10.79		-11.92	18.84		-12.32	19.79

Table 8. Average results for q-TSP-200-x instance in single-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-200-1	1066186	932422	-12.55	368.90	915860	-14.10	375.56	916838	-14.01	375.87	916838	-14.01	379.76
TSP-200-2	1062958	954427	-10.21	366.95	937735	-11.78	378.62	944975	-11.10	376.95	938650	-11.69	379.60
TSP-200-3	1071636	986014	-7.99	366.21	963599	-10.08	376.15	965917	-9.87	379.16	962012	-10.23	375.26
TSP-200-4	1057787	925559	-12.50	367.02	909991	-13.97	374.92	912294	-13.75	379.02	910578	-13.92	378.69
TSP-200-5	1038257	912686	-12.09	365.48	904108	-12.92	378.52	904176	-12.91	375.30	903773	-12.95	376.35
TSP-200-6	1025893	898895	-12.38	365.66	885317	-13.70	378.90	891344	-13.12	377.00	885603	-13.67	377.11
TSP-200-7	1049443	934193	-10.98	369.71	918955	-12.43	376.19	919909	-12.34	377.63	919909	-12.34	377.74
TSP-200-8	1069744	942862	-11.86	369.78	933926	-12.70	374.56	939799	-12.15	377.08	935149	-12.58	379.71
TSP-200-9	1051576	940649	-10.55	367.88	923359	-12.19	375.29	926328	-11.91	378.28	926328	-11.91	377.09
TSP-200-10	1056032	940673	-10.92	365.30	938821	-11.10	376.04	936899	-11.28	378.14	931138	-11.83	379.92
TSP-200-11	1074345	951744	-11.41	366.17	937992	-12.69	376.97	938700	-12.63	376.46	930173	-13.42	376.51
TSP-200-12	1047153	925457	-11.62	366.77	923479	-11.81	375.31	917742	-12.36	377.16	916272	-12.50	378.51
TSP-200-13	1050723	916100	-12.81	369.11	914837	-12.93	377.01	906991	-13.68	375.08	906991	-13.68	378.33
TSP-200-14	1044901	931711	-10.83	365.08	920265	-11.93	377.56	920068	-11.95	379.92	912006	-12.72	377.70
TSP-200-15	1051583	942546	-10.37	365.22	933916	-11.19	375.11	926928	-11.85	375.84	924515	-12.08	378.49
TSP-200-16	1046666	930359	-11.11	365.84	924628	-11.66	374.59	927219	-11.41	375.53	917122	-12.38	378.33
TSP-200-17	1054887	948491	-10.09	368.25	926230	-12.20	375.48	921009	-12.69	376.86	916958	-13.08	375.89
TSP-200-18	1062649	942866	-11.27	368.66	937974	-11.73	375.59	935843	-11.93	375.99	930747	-12.41	375.64
TSP-200-19	1032453	915980	-11.28	368.24	916077	-11.27	376.12	908604	-12.00	377.45	907400	-12.11	380.00
TSP-200-20	994281	869683	-12.53	367.25	867280	-12.77	376.54	861844	-13.32	376.70	859655	-13.54	375.86
Aver			-11.27	367.17		-12.26	376.25		-12.31	377.07		-12.65	377.82

Table 9. Average results for q-TSP-30-x instance in multi-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-30-1	22397	19949	-10.93	0.03	19402	-13.37	0.04	19402	-13.37	0.08	19402	-13.37	0.12
TSP-30-2	22370	20163	-9.87	0.03	19812	-11.43	0.04	19811	-11.44	0.08	19811	-11.44	0.12
TSP-30-3	20412	17942	-12.10	0.03	17745	-13.07	0.06	17745	-13.07	0.08	17745	-13.07	0.1
TSP-30-4	23811	21245	-10.78	0.05	21093	-11.41	0.06	21078	-11.48	0.09	21067	-11.52	0.09
TSP-30-5	21664	19363	-10.62	0.04	19136	-11.67	0.06	19094	-11.86	0.08	19094	-11.86	0.12
TSP-30-6	21021	19235	-8.50	0.04	18940	-9.90	0.05	18940	-9.90	0.08	18938	-9.91	0.12
TSP-30-7	23784	21535	-9.46	0.03	21081	-11.36	0.05	21081	-11.36	0.08	21081	-11.36	0.12
TSP-30-8	23033	20631	-10.43	0.03	20024	-13.06	0.06	20024	-13.06	0.08	20024	-13.06	0.09
TSP-30-9	21331	19458	-8.78	0.04	19050	-10.69	0.05	19050	-10.69	0.08	19050	-10.69	0.11
TSP-30-10	21867	19425	-11.17	0.03	19167	-12.35	0.05	19167	-12.35	0.09	19167	-12.35	0.13
TSP-30-11	20048	18301	-8.71	0.04	17645	-11.99	0.06	17645	-11.99	0.08	17643	-12.00	0.11
TSP-30-12	18507	16921	-8.57	0.04	16638	-10.10	0.06	16638	-10.10	0.09	16581	-10.41	0.11
TSP-30-13	22476	20547	-8.58	0.04	19719	-12.27	0.04	19743	-12.16	0.09	19718	-12.27	0.1
TSP-30-14	21060	19495	-7.43	0.03	18833	-10.57	0.04	18833	-10.57	0.08	18833	-10.57	0.13
TSP-30-15	22085	20091	-9.03	0.03	19581	-11.34	0.06	19578	-11.35	0.09	19577	-11.36	0.11
TSP-30-16	20826	18892	-9.29	0.04	18663	-10.39	0.05	18660	-10.40	0.08	18660	-10.40	0.12
TSP-30-17	22753	19748	-13.21	0.03	19388	-14.79	0.05	19388	-14.79	0.08	19388	-14.79	0.12
TSP-30-18	20686	18788	-9.18	0.04	18505	-10.54	0.05	18505	-10.54	0.09	18505	-10.54	0.11
TSP-30-19	22352	19194	-14.13	0.04	19135	-14.39	0.06	19045	-14.80	0.09	19011	-14.95	0.12
TSP-30-20	21647	19657	-9.19	0.03	19241	-11.11	0.05	19241	-11.11	0.08	19241	-11.11	0.12
Aver			-10.00	0.036		-11.79	0.052		-11.82	0.084		-11.85	0.114

Table 10. Average results for q-TSP-50-x instance in multi-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-50-1	65007	59789	-8.03	3.13	58992	-9.25	4.18	58543	-9.94	5.22	58992	-9.25	5.30
TSP-50-2	66861	60703	-9.21	3.25	59301	-11.31	4.19	58764	-12.11	5.14	59301	-11.31	5.22
TSP-50-3	63723	57928	-9.09	3.32	56460	-11.40	4.26	56507	-11.32	5.26	56460	-11.40	5.14
TSP-50-4	69108	62074	-10.18	3.27	61014	-11.71	4.18	61063	-11.64	5.23	61014	-11.71	5.18
TSP-50-5	64880	60865	-6.19	3.16	59726	-7.94	4.31	59795	-7.84	5.28	59726	-7.94	5.16
TSP-50-6	68108	61450	-9.78	3.20	59466	-12.69	4.35	59466	-12.69	5.28	59466	-12.69	5.18
TSP-50-7	70956	61348	-13.54	3.23	58348	-17.77	4.29	58356	-17.76	5.27	58348	-17.77	5.22
TSP-50-8	68834	63088	-8.35	3.35	61207	-11.08	4.20	61305	-10.94	5.13	61207	-11.08	5.24
TSP-50-9	69326	62014	-10.55	3.16	60338	-12.96	4.25	60517	-12.71	5.14	60338	-12.96	5.23
TSP-50-10	67715	57914	-14.47	3.32	57423	-15.20	4.14	57374	-15.27	5.19	57423	-15.20	5.32
TSP-50-11	63051	53652	-14.91	3.27	53306	-15.46	4.33	52860	-16.16	5.24	53306	-15.46	5.24
TSP-50-12	61257	54530	-10.98	3.21	54234	-11.46	4.32	52922	-13.61	5.27	54234	-11.46	5.34
TSP-50-13	66835	59674	-10.71	3.16	59383	-11.15	4.31	58980	-11.75	5.21	59383	-11.15	5.27
TSP-50-14	67593	60759	-10.11	3.22	58898	-12.86	4.18	58634	-13.25	5.31	58898	-12.86	5.34
TSP-50-15	67333	58442	-13.20	3.23	57098	-15.20	4.26	56635	-15.89	5.29	57098	-15.20	5.18
TSP-50-16	70420	59402	-15.65	3.15	58449	-17.00	4.13	57984	-17.66	5.34	58449	-17.00	5.28
TSP-50-17	68602	60509	-11.80	3.26	60167	-12.30	4.22	59820	-12.80	5.24	60167	-12.30	5.19
TSP-50-18	64936	60462	-6.89	3.17	60297	-7.14	4.19	59822	-7.88	5.19	60297	-7.14	5.27
TSP-50-19	66249	60651	-8.45	3.21	58497	-11.70	4.16	58666	-11.45	5.14	58497	-11.70	5.28
TSP-50-20	67387	58519	-13.16	3.25	58102	-13.78	4.16	57596	-14.53	5.26	58102	-13.78	5.14
Aver			-10.76	3.23		-12.47	4.23		-12.86	5.23		-12.47	5.23

Table 11. Average results for q-TSP-100-x instance in multi-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-100-1	254367	229313	-9.85	105.09	229926	-9.61	128.37	227960	-10.38	227.95	227060	-10.74	226.82
TSP-100-2	273942	247520	-9.65	104.99	246762	-9.92	128.57	245285	-10.46	227.69	244086	-10.90	227.10
TSP-100-3	264618	235326	-11.07	106.49	236153	-10.76	128.29	234018	-11.56	229.79	232516	-12.13	226.57
TSP-100-4	264176	232994	-11.80	107.09	232086	-12.15	130.27	230229	-12.85	225.92	229275	-13.21	229.11
TSP-100-5	269213	235668	-12.46	105.40	235220	-12.63	125.57	234974	-12.72	226.26	233222	-13.37	228.24
TSP-100-6	259488	232003	-10.59	106.88	233432	-10.04	128.76	229941	-11.39	224.99	229402	-11.59	229.16
TSP-100-7	268516	238569	-11.15	106.52	235907	-12.14	125.74	236076	-12.08	230.24	233914	-12.89	230.19
TSP-100-8	266177	232567	-12.63	104.25	232273	-12.74	124.99	233214	-12.38	228.36	230559	-13.38	230.45
TSP-100-9	266762	242423	-9.12	106.27	238839	-10.47	128.11	236776	-11.24	227.29	236776	-11.24	225.46
TSP-100-10	267537	233891	-12.58	105.54	233124	-12.86	127.11	231577	-13.44	228.32	231288	-13.55	225.12
TSP-100-11	260790	237922	-8.77	107.34	235817	-9.58	127.16	234600	-10.04	227.71	234044	-10.26	228.68
TSP-100-12	259433	231504	-10.77	104.23	230068	-11.32	128.46	229449	-11.56	228.37	229276	-11.62	224.83
TSP-100-13	252283	222016	-12.00	105.80	220506	-12.60	129.15	217559	-13.76	227.71	217503	-13.79	227.59
TSP-100-14	284030	242532	-14.61	105.67	242509	-14.62	126.47	240937	-15.17	228.84	240305	-15.39	227.62
TSP-100-15	247662	219652	-11.31	105.79	218820	-11.65	128.46	221707	-10.48	227.57	217038	-12.37	229.73
TSP-100-16	279004	249739	-10.49	106.84	248109	-11.07	126.89	244459	-12.38	230.58	244459	-12.38	227.33
TSP-100-17	264287	231400	-12.44	105.32	230987	-12.60	129.61	229537	-13.15	225.63	229190	-13.28	226.74
TSP-100-18	251266	223684	-10.98	106.89	223126	-11.20	129.55	219862	-12.50	224.91	219862	-12.50	228.52
TSP-100-19	262380	236906	-9.71	105.83	236590	-9.83	125.87	234975	-10.44	224.93	233915	-10.85	228.97
TSP-100-20	256873	232339	-9.55	104.35	230850	-10.13	128.15	230233	-10.37	224.64	228810	-10.92	227.55
Aver			-11.08	105.83		-11.40	127.78		-11.92	227.38		-12.32	227.79

Table 12. Average results for q-TSP-200-x instance in multi-start

Instances	*UB*	VND			VNS			GVNS			GVNS-AM		
		Best.Sol	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*	*Best.Sol*	*Gap* [%]	*Time*
TSP-200-1	1066186	932422	-12.55	3689.01	915860	-14.10	3737.99	916838	-14.01	4173.61	916838	-14.01	4181.12
TSP-200-2	1062958	954427	-10.21	3669.49	937735	-11.78	3701.81	944975	-11.10	4181.87	938650	-11.69	4174.64
TSP-200-3	1071636	986014	-7.99	3662.08	963599	-10.08	3744.09	965917	-9.87	4175.25	962012	-10.23	4182.88
TSP-200-4	1057787	925559	-12.50	3670.20	909991	-13.97	3698.24	912294	-13.75	4165.58	910578	-13.92	4177.68
TSP-200-5	1038257	912686	-12.09	3654.82	904108	-12.92	3733.13	904176	-12.91	4166.58	903773	-12.95	4175.13
TSP-200-6	1025893	898895	-12.38	3656.60	885317	-13.70	3698.76	891344	-13.12	4171.21	885603	-13.67	4180.33
TSP-200-7	1049443	934193	-10.98	3697.10	918955	-12.43	3736.77	919909	-12.34	4178.84	919909	-12.34	4181.75
TSP-200-8	1069744	942862	-11.86	3697.81	933926	-12.70	3731.84	939799	-12.15	4180.56	935149	-12.58	4184.20
TSP-200-9	1051576	940649	-10.55	3678.76	923359	-12.19	3696.96	926328	-11.91	4179.76	926328	-11.91	4162.13
TSP-200-10	1056032	940673	-10.92	3652.99	938821	-11.10	3682.07	936899	-11.28	4169.23	931138	-11.83	4181.45
TSP-200-11	1074345	951744	-11.41	3661.74	937992	-12.69	3735.10	938700	-12.63	4174.05	930173	-13.42	4175.80
TSP-200-12	1047153	925457	-11.62	3667.66	923479	-11.81	3749.77	917742	-12.36	4164.13	916272	-12.50	4184.23
TSP-200-13	1050723	916100	-12.81	3691.06	914837	-12.93	3692.36	906991	-13.68	4164.61	906991	-13.68	4173.90
TSP-200-14	1044901	931711	-10.83	3650.77	920265	-11.93	3724.10	920068	-11.95	4165.16	912006	-12.72	4172.83
TSP-200-15	1051583	942546	-10.37	3652.15	933916	-11.19	3702.62	926928	-11.85	4177.27	924515	-12.08	4180.01
TSP-200-16	1046666	930359	-11.11	3658.45	924628	-11.66	3739.08	927219	-11.41	4173.18	917122	-12.38	4167.21
TSP-200-17	1054887	948491	-10.09	3682.46	926230	-12.20	3733.11	921009	-12.69	4166.36	916958	-13.08	4173.24
TSP-200-18	1062649	942866	-11.27	3686.59	937974	-11.73	3677.56	935843	-11.93	4173.17	930747	-12.41	4182.24
TSP-200-19	1032453	915980	-11.28	3682.39	916077	-11.27	3741.71	908604	-12.00	4165.42	907400	-12.11	4174.95
TSP-200-20	994281	869683	-12.53	3672.55	867280	-12.77	3753.51	861844	-13.32	4163.35	859655	-13.54	4180.99
Aver			-11.27	3671.73		-12.26	3720.53		-12.31	4171.46		-12.65	4177.33

REFERENCES

[1] H.B. Ban, "A GRASP+VND algorithm for the multiple traveling repairman problem with distance constraints", *Journal of Computer Science and Cybernetics*, vol.33, no.3, 2017, pp. 272–288.

[2] R. Bermudez, M. H. Cole, "A genetic algorithm approach to door assignments in breakbulk

terminals", University of Arkansas, Mack-Blackwell National Rural Transportation, 2001, Study Center, Fayetteville.

[3] C. Blum, A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", *ACN Computing Surveys*, vol. 35, no. 3, 2003, pp. 268–308.

[4] E. Cela, V. G. Deineko, and G. J. Woeginger, "The multi-stripe travelling salesman problem", *J. Annals of Operations Research,* vol. 259, 2017, pp. 21–34.

[5] N. Christofides, "Worst case analysis of a new heuristic for the Travelling salesman problem", Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[6] T. A. Feo and M.G.C. Resende, "Greedy randomized adaptive search procedures", *Journal of Global Optimization*, 1995, pp. 109–133.

[7] G. Finke, E. B. Rainer, F. Rendl, "Quadratic assignment problems", *J. North-Holland Mathematics Studies*, vol. 132, 1987, pp. 61–82.

[8] A.M. Geoffrion, G.W. Graves, "Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/LP approach", *J. Oper Res*, vol. 24, no. 4, 1976, pp.595–610.

[9] M. Grotschel, "Discrete mathematics in manufacturing", *Proc. ICIAM on SIAM*, Robert E. O'Malley (ed.), 1992, pp. 119–145.

[10] D. S. Johnson, and L. A. McGeoch, "The traveling salesman problem: A Case Study in Local Optimization in Local Search in Combinatorial Optimization", E. Aarts and J. K. Lenstra, eds., pp. 215-310.

[11] A. Mason, M. Ronnqvist, "Solution methods for the balancing of jet turbines, *J. Comput Oper Res",* vol. 24, no. 2, 1997, pp. 153–167.

[12] O. Martin, S. W. Otto, and E.W. Felten, "Large-step Markov chains for the travelling salesman problem", *J. Complex Systems,* vol. 5, no. 3, 1991, pp. 299–326.

[13] I. Mathlouthi, M. Gendreau, J. Y. Potvin, *A metaheuristic based on tabu search for solving a technician routing and scheduling problem,* 2018.

[14] N. Mladenovic, P. Hansen, "Variable neighborhood search", *Computers & Operations Research,* vol. 24, no. 11, 1997, pp.1097–1100.

[15] I. Ugi, J. Bauer, J. Brandt, J. Friedrich, J. Gasteige, C. Jochum C, W. Schubert, "New Fields of Application for Computers in Chemistry", Angew Chem, Vol. 91, No. 2, 1979, pp. 111-123.

[16] A. T. Phillips, J. B. Rosen, " A quadratic assignment formulation of the molecular conformation problem," *J Glob Optim,* vol. 4, pp. 229–241, 1994. https://doi.org/10.1007/BF01096724

[17] K. Ruland, "Polyhedral Solution to the Pickup and Delivery Problem," Ph.D. Thesis, Washington University, Saint Louis, MO, 1995.

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

Received on January 07, 2020
Revised on April 06, 2020