IOCA 2021

Proceedings

A Fast Algorithm for Euclidean Bounded Single-Depot Multiple Traveling Salesman Problem[†]

Víctor Pacheco-Valencia [1], Nodari Vakhania [1*], José Alberto Hernández [2] and Juan Carlos Hernández-Gómez [3]

[1] Centro de Investigación en Ciencias UAEMor; Universidad Autónoma del Estado de Morelos, Av. Universidad 1001, Col.Chamilpa, post code 62209, Cuernavaca, Morelos, México.
[2] Facultad de Contaduría, Administración e Informática UAEMor; Universidad Autónoma del Estado de Morelos, Av. Universidad 1001, Col.Chamilpa, post code 62209, Cuernavaca, Morelos, México.
[3] Facultad de Matemáticas UAGro; Universidad Autónoma de Guerrero, Carlos E. Adame No.54, Col.Garita, post code 39650, Acapulco, Guerrero, México.
* Correspondence: nodari@uaem.mx

† 1st International Online Conference on Algorithms (IOCA 2021), https://ioca2021.sciforum.net/, September 27 - October 10/2021.

Abstract: The Multiple Traveling Salesman Problem (MTSP) is a combinatorial optimization problem that can model some real-life problems. There are given $n+1$ objects that are commonly referred to as cities, among which there is one distinct city called depot, and k additional objects commonly referred to as salesman. Each salesman has to build its own tour that starts from the depot, ends also in depot and visits only once one or more other cities. Visiting city j from city i implies a cost c_{ij}. The cost of a tour is the sum of the individual costs of each pair of cities from that tour. The aim is to minimize the total cost of all k tours. Here we consider the two-dimensional Euclidean version of the problem and impose lower and upper bounds on the minimum and maximum number of cities in a tour suggesting a 3-phase heuristic algorithm for that version. At the first phase the whole set of cities is partitioned into k disjoint subsets, at the second phase a feasible tour for each of these subsets is constructed, and at phase 3 these feasible tours are iteratively improved. We report preliminary experimental results for the 22 benchmark instances. The approximation gap provided by the proposed heuristic is comparable to the state of the art results, whereas it is much faster than earlier known state-of-the-art algorithms.

Keywords: travelling salesman problem; algorithm; time complexity

check for updates

Citation: Pacheco-Valencia, V; Vakhania N., Hernández, J.A.; and Hernández-Gómez J.C. A Fast Algorithm for Euclidean Bounded Single-Depot Multiple Traveling Salesman Problem. *Algorithms* **2021**, *1*, 0. https://doi.org/

Published:

1. Introduction

Multiple Traveling Salesman Problem (MTSP) is a generalization of a well-known Traveling Salesman Problem (TSP) that reflects better the needs in some practical circumstances. Instead of a single salesman, in MTSP there are $k \geq 1$ salesmen and one distinguished point called *depot*. An independent tour starting and ending at the depot is to be constructed for each salesman, and the objective is to minimize the total incurred distance/time.

MTSP is a relaxation of another widely studied Capacitated Vehicle Routing Problem (CVRP), also known as Capacitated MTSP (CMTSP), where each salesman drives a vehicle with a limited capacity and each client has some demand so that the sum of the demands of the clients from a tour can not exceed the capacity of the vehicle that is assigned to that tour (a vehicle has a limited fuel/energy capacity).

The MTSP problem can be described as follows. We are given an undirected weighted complete graph $G = (V, E)$ with $n^2 - n$ edges such that the set V of $n+1$ vertices contains one distinguished vertex $d = n+1$ called *depot* and a non-negative weight $\mathrm{w}(i, j)$ is associated with each edge $(i, j) \in E$. Each of the n vertices from set $V \setminus \{d\}$ represents one of the n clients, and weight $\mathrm{w}(i, j)$ sets the time/distance between vertices i and j.

An integer k represents the number of different *feasible tours* in a solution, a sequence of vertices which starts and ends at the depot and contains at least one more vertex exactly once (so that the salesman visits every vertex from the tour only once except the depot which it visits twice). A *feasible solution* is a union of exactly k feasible tours.

Given a feasible tour T^j of salesman j

$$T^j = (d, i_1^j, i_2^j, \cdots, i_{m_j-1}^j, i_{m_j}^j, d), \tag{1}$$

the *cost of that tour* is

$$\mathrm{C}(T^j) = \mathrm{w}(d, i_1^j) + \mathrm{w}(i_1^j, i_2^j) + \cdots + \mathrm{w}(i_{m_j-1}^j, i_{m_j}^j) + \mathrm{w}(i_{m_j}^j, d). \tag{2}$$

The *cost a feasible solution* $T = \{T^1, T^2, \cdots, T^k\}$ is

$$\mathrm{C}(T) = \mathrm{C}(T^1) + \mathrm{C}(T^2) + \cdots + \mathrm{C}(T^k). \tag{3}$$

The objective is to find an optimal solution, a feasible solution with the minimum cost, $min_T\, \mathrm{C}(T)$.

In practical terms, it is clear that an optimal solution provides higher profits for an involved company and also a higher customer satisfaction. Besides, the vehicles and machines involved in the customer service process have less wear and tear causing less contamination.

Bektas[1] in 2006 described some variants for an MTSP with more than one depot, with an upper limit on the number of clients in a tour and a permissible time window for the salesman to visit each client on its tour. In some settings, the number of salesmen is not fixed in advanced, the minimal number of salesmen is 1 and there is a penalty for introducing each new salesmen in the solution. Bektas[1] and, Kara & Bektas[2] present alternative integer programming formulations for MTSP. More recently, Cheikhrouhou & Khoufi [3] presented another survey focusing on some related real-life problems including Unmanned Aerial Vehicle problems.

In this paper, we focus on a version of MTSP known as Bounded Single-Depot Multiple Traveling Salesman Problem (Bounded SD-MTSP) which is an extension of MTSP with a restriction on the number of clients in each tour, which is allowed to be any magnitude from a priory given range $[m_{\min}, m_{\max}]$. We propose a three-phase deterministic solution method that we call *PCI*-algorithm (Partition, Construction and Improvement algorithm). *PCI*-algorithm uses some ideas from [4] and [5] and is designed to construct a feasible solution quickly and then improve it. The algorithm from [4] for problem MTSP works in two basic phases, at phase 1 the whole set of vertices is partitioned into k disjoint subsets based on the girding polygon and specially formed auxiliary separator edges. At phase 2, a feasible tour for each subset from the formed partition is constructed. The algorithm for TSP from [5] has three phases. At phase 1 an (infeasible) partial tour based on the girding polygon from [4] is constructed. At phase 2 the insertion of yet undistributed vertices are carried out until a feasible solution is obtained. At phase 3 current feasible solution is improved iteratively by altering the current position of some vertices from the solution. At it was shown experimentally, this algorithm consumes a little computer memory and is considerably faster than other state-of-the-art algorithms.

As in [4], *PCI*-algorithm initially partitions the set of vertices in k disjoint subsets at phase 1. Then at phase 2 it constructs the initial k tours using the algorithm from [5] for the traveling salesman problem. The feasible solution of phase 2 is further improved at phase 3: iteratively, a vertex from a tour is moved from its current position to another specially determined position within the same or another tour so that the resultant solution remains feasible. The destiny vertex and its new position are selected so that the accomplished rearrangement provides the maximum decrease of the current cost.

We report preliminary experimental results for the 22 benchmark instances. The approximation gap provided by the proposed heuristic is comparable to the state of the art results, whereas it is much faster than earlier known state-of-the-art algorithms.

2. Methods

From here on we concentrate our attention to the bounded version of MTSP (BMTSP) in which the given lower and upper bounds restrict the number of vertices in a tour.

This section describes our first heuristic algorithm for the bounded version BMTSP of the problem with k salesmen. It consists of three phases. At phase 1 a partition of set $V \setminus \{d\}$ into k (disjoint) subsets is carried out. This reduces the problem into k sub-problems, each of which can be seen as an instance of TSP. At phase 2, for each of the partitions (sub-problems) the heuristic from [5] is applied to construct a feasible tour. This results in a feasible solution for the problem BMTSP. At phase 3, an improvement of the solution of phase 2 is carried out.

2.1. Phase 1

We describe now phase 1 specifying how the partition of set $V \setminus \{d\}$ in k subsets $V_1, V_2, \cdots, V_k$ is performed (each of them having the cardinality within the range $[m_{\min}, m_{\max}]$). Phase 1 consists of the initial assignments, stage 1 and stage 2. Below we describe how initial assignments are performed.

Recall that we consider the Euclidean version of the problem, and hence we deal with the 2-dimensional Euclidean plane. We first define an auxiliary point c on that plane, that we call the *centroid*. The x and y coordinates x_c and y_c of this point are obtained by averaging the x and the y coordinates of the n vertices from $V \setminus \{d\}$, i.e., $x_c = \frac{\sum_{i=1}^{n} x_i}{n}$ and $y_c = \frac{\sum_{i=1}^{n} y_i}{n}$.

Let $i'_1 \in V \setminus \{d\}$ be any furthest vertex from depot d. We determine $k-1$ additional auxiliary points i'_j, $j = 2, \dots, k$ so that each of these points are at the same distance from centroid c and the lines from point c to each of i'_j divide the plane into k symmetric areas, i.e., the angle between each two neighboring lines is $2\pi/k$. Procedure $COORDINATES(c, i'_1, j, k)$ below specifies how the coordinates $x_{i'_j}$ and $y_{i'_j}$ of each point i'_j are calculated.

Procedure 1: $COORDINATES(c, i'_1, j, k)$

```
1  $\theta := \arctan\left(\frac{y_{i'_1} - y_c}{x_{i'_1} - x_c}\right)$
2  if $\theta + (j-1)\frac{2\pi}{k} < \pi$ then
3      $x_{i'_j} := x_c + \mathrm{w}(c, i'_1)\cos\left(\theta + (j-1)\frac{2\pi}{k}\right)$
4      $y_{i'_j} := y_c + \mathrm{w}(c, i'_1)\sin\left(\theta + (j-1)\frac{2\pi}{k}\right)$
5  else
6      $j' := 1$
7      while $\theta + (j'-1)\frac{2\pi}{k} < \pi$ do
8          $j' := j' + 1$
9      $x_{i'_j} := x_c + \mathrm{w}(c, i'_1)\cos\left(\theta - (j - j' + 1)\frac{2\pi}{k}\right)$
10     $y_{i'_j} := y_c + \mathrm{w}(c, i'_1)\sin\left(\theta - (j - j' + 1)\frac{2\pi}{k}\right)$
11 return $i'_j$
```

The two stages of Phase 1 use another auxiliary procedure $CLOSEST(B, A, out)$. Roughly, set A consists of the vertices from subset V_i, one of the subsets from the current partially formed partition $V_1, \cdots, V_k$. Set B consists of the vertices which are not assigned to any of the partially formed tour, i.e., $B = V \setminus \{\cup_j \{V_j\} \cup \{d\}\}$. $CLOSEST(B, A, out)$ outputs a vertex $b \in B$ that is closest to a vertex from set A, if the parameter $out =$ "*vertex*"; if $out =$ "*distance*", then the procedure returns the distance between the closest two vertices of sets A and B.

Procedure 2: $CLOSEST(B, A, out)$

```
input: $A = \{a_1, a_2, \cdots, a_{|A|}\}$   $B = \{b_1, b_2, \cdots, b_{|B|}\}$
b := 0
d := ∞
for $i := 1$ to $|B|$ do
    for $j := 1$ to $|A|$ do
        if $\sqrt{(x_{a_j} - x_{b_i})^2 + (y_{a_j} - y_{b_i})^2} < d$ then
            $b := b_i$
            $d := \sqrt{(x_{a_j} - x_{b_i})^2 + (y_{a_j} - y_{b_i})^2}$
if $out$ = "vertex" then
    return $b$                                  // return the vertex
if $out$ = "distance" then
    return $d$                                  // return the distance
```

Now, stage 1, initially sets $V_j := \emptyset$ and $B := V \setminus \{d\}$. Iteratively, invoking procedure $CLOSEST(B, V_j, \text{"vertex"})$ $m_{\min}$ times every subset V_j is augmented at each iteration with a new vertex by $V_j := V_j \cup CLOSEST(B, V_j, \text{"vertex"})$, where that vertex is eliminated from the current set B. As a result, we are left with the feasible subsets V_j, i.e., ones with exactly $m_{\min}$ vertices.

If the above formed feasible partition $V_1, \cdots, V_k$ is not complete, i.e., it does not contain all the vertices from set $V \setminus \{d\}$ and hence the corresponding solution is not feasible, then phase 1 enters the second stage which augments the cardinality of some of the subsets and creates the complete feasible partition, that is again denoted by $V_1, \cdots, V_k$. Iteratively, procedure $CLOSEST(B, V_j, \text{"vertex"})$ is invoked for each subset V_ι with $|V_\iota| < m_{\max}$ and a vertex with the shortest distance from the corresponding subset is added to the former subset V_ι. In other words, $V_j := V_j \cup CLOSEST(B, V_j, \text{"vertex"})$, where the distance of the vertex $CLOSEST(B, V_j, \text{"distance"})$ to the closest vertex of set V_j is the minimum possible among all subsets V_ι with $|V_\iota| < m_{\max}$ (again, set B is updated as in stage 1). Phase 1 halts with a complete feasible partition, again denoted by $V_1, \cdots, V_k$.

2.2. Phase 2

At phase 2 a feasible tour for each of the subsets $V_j \cup \{d\}$ of phase 1 is constructed by merely invoking the algorithm *CII* from [5].

2.3. Phase 3

Phase 3 applies local rearrangement to the solution of phase 2 to improve that solution. It invokes procedure 4 that repeatedly selects a vertex from tour T^j and alters its position within that tour or this vertex is relocated to another tour $T^{j'}$. The exchange is accomplished if it yields the decrease of the total cost; among all such exchanges, one which yields the maximum decrease is accomplished so that the resultant solution remains feasible.

For the convenience of the presentation, at phase 3, we represent a feasible tour T^j as $T^j = (i^j_0, i^j_1, \cdots, i^j_{m_j}, i^j_{m_j+1})$, where $i^j_0 = i^j_{m_j+1} = d$. Phase 3 repeatedly invokes procedure 4, $RELOCATE(T^1, \cdots, T^k)$ that relocates a selected vertex from one tour to another. For each vertex $i^j_l \in T^j$, $j = 1 \ldots, k$, the gain after removing that vertex from tour T^j is $G(i^j_l) = \mathrm{w}(i^j_{l-1}, i^j_{l+1}) - \mathrm{w}(i^j_{l-1}, i^j_l) - \mathrm{w}(i^j_l, i^j_{l+1})$, here $j \in [1, k]$ and $l \in [1, m_{j+1}]$ (note that by the triangle inequality, $G(i^j_l) \leq 0$). Likewise, the cost of the insertion of vertex i^j_l in between vertices $i^{j'}_{l'-1}$ and $i^{j'}_{l'}$ in tour $T^{j'}$ is $C(i^j_l, j', l') := -w(i^{j'}_{l'-1}, i^{j'}_{l'}) + w(i^{j'}_{l'-1}, i^j_l) + w(i^j_l, i^{j'}_{l'})$ (here $j' \in [1, k]$ and $l' \in [1, m_{j'+1}]$, note that $j = j'$ is possible). All possible exchanges are verified and an exchange that yields the minimum $G(i^j_l) + C(i^j_l, j', l')$ is carried out, given that $G(i^j_l) + C(i^j_l, j', l') < 0$. This kind of an exchange is iteratively performed at phase 3 until

Procedure 3: $PHASE\_1(i'_1, c, V, k, m_{\min}, m_{\max})$

```
for $j := 1$ to $k$ do
    $V_j := \emptyset$
$B := V \setminus \{d\}$                                  // vertices not included in any subset $V_j$
// Stage 1:
$V_1 := V_1 \cup CLOSEST(B, \{i'_1\}, \text{"vertex"})$   // Add the first vertex to each subset $V_j$
for $j := 2$ to $k$ do
    $i'_j := COORDINATES(c, i'_1, j, k)$
    $b := CLOSEST(B, \{i'_j\}, \text{"vertex"})$
    $V_j := V_j \cup \{b\}$                                                        // $b \in B$
    $B := B \setminus \{b\}$
                                           // Add $m_{\min} - 1$ vertex to each subset $V_j$
for $m := 2$ to $m_{\min}$ do
    for $j := 1$ to $k$ do
        $b := CLOSEST(B, V_j, \text{"vertex"})$
        $V_j := V_j \cup \{b\}$
        $B := B \setminus \{b\}$
// Stage 2: If the above formed feasible partition is not complete
while $|B| > 0$ do
    $d' := \infty$
    for $j := 1$ to $k$ do
        if $|V_j| < m_{\max}$ then
            $d := CLOSEST(B, V_j, \text{"distance"})$
        if $d < d'$ then
            $d' := d$
            $b' := CLOSEST(B, V_j, \text{"vertex"})$
            $j' := j$
    $V_{j'} := V_{j'} \cup \{b'\}$
    $B := B \setminus \{b'\}$
return $V_1, \cdots, V_k$
```

the latter sum remains negative. Phase 3 halts with the solution of the previous iteration when the minimum $G(i_l^j) + C(i_l^j, j', l')$ becomes non-negative.

Procedure 4: $RELOCATE(T^1, \cdots, T^k)$

```
input: $T^j = \{i_0^j, i_1^j, \cdots, i_{m_j}^j, i_{m_j+1}^j\}$ where $i_0^j = i_{m_j+1}^j = d$
$G' := \infty$
for $j := 1$ to $k$ do
    if $m_j > m_{\min}$ then
        for $l := 1$ to $m_j + 1$ do
            $G := w(i_{l-1}^j, i_{l+1}^j) - w(i_{l-1}^j, i_l^j) - w(i_l^j, i_{l+1}^j)$    // The gain after removing $i_l^j$ from $T^j$
            for $j' := 1$ to $k$ do
                if $m_{j'} < m_{\max}$ then
                    for $l' := 1$ to $m_{j'} + 1$ do
                        if $c_{i'-1}^{j'} \neq c_i^j \wedge c_{i'}^{j'} \neq c_i^j$ then
                            $C := -w(i_{l'-1}^{j'}, i_{l'}^{j'}) + w(i_{l'-1}^{j'}, i_l^j) + w(i_l^j, i_{l'}^{j'})$
                               // The cost of the insertion of vertex $i_l^j$ between $i_{l'-1}^{j'}$ and
                              $i_{l'}^{j'}$ in tour $T^{j'}$
                        else
                            $C := \infty$
                        if $G + C < G'$ then
                            $p := i_l^j$
                            $q := i_{l'}^{j'}$
                            $G' := G + C$
return $G', p, q$
```

Procedure 5: $PHASE\_3(T^1, \cdots, T^k)$

```
$stop := false$
while $stop = false$ do
    $G', i_l^j, i_{l'}^{j'} := RELOCATE(T^1, \cdots, T^k)$
    if $G' < 0$ then
        $i := i_l^j$
        remove vertex $i$ from the tour $T^j$
        insert vertex $i$ between vertices $i_{l'-1}^{j'}$ and $i_{l'}^{j'}$ in tour $T^{j'}$
    else
        $stop := true$
return $T^1, \cdots, T^k$
```

3. Implementation and Results

PCI-algorithm was coded in C++ and compiled on a server with processor 2x Intel Xeon E5-2650 0 @ 2.8 GHz, with 32 GB in RAM and Debian GNU/Linux 10 (buster) operating system.

To evaluate the performance of our algorithm, we used the instances proposed by Junjie & Dingwei[7] in 2006, who chose six TSP instances from TSPLIB[15] (*pr76*, *pr152*, *pr226*, *pr299*, *pr439* and *pr1002*), and Necula *et al.*[8] in 2015, who chose other 4 TSP instances from TSPLIB[15] (*eil51*, *berlin52*, *eil76* and *rat99*). These instances were transformed into 22 BMTSP instances for by setting the corresponding first vertices as the depot and the rest of the vertices as the n clients. A positive integer was chosen for k (the number of salesmen) and the minimum and the maximum number of clients is each a tour.

The main results are shown in table 1. These tables specify the time and the cost of the solutions obtained by *PCI*-algorithm and the Best Known Solutions (BKS) for the 6 instances from [7,9–14]. We also report our results for the 16 instances from [8] where the results are estimated based on the solutions provided by CPLEX [17].

Table 1. Comparison of the Best Know Solutions (BKS) and *PCI*-algorithm.

Instance					***PCI*-algorithm**			**Algorithm**		
Name	n	k	$m_{\min}$	$m_{\max}$	$C(PCI)$	$Error_{PCI}$	$Time_{PCI}$	$C(BKS)$	$Time_{BKS}$	**Name**
pr76.tsp	75	5	1	20	173,820	31%	63 ms.	132,784	5 s.	GELS-GA[10]
pr152.tsp	151	5	1	40	135,835	29%	246 ms.	105,205	8 s.	GELS-GA[10]
pr226.tsp	225	5	1	50	156,015	5%	222 ms.	148,051	76.6 s.	GAL[11]
pr299.tsp	298	5	1	70	75,064	3%	816 ms.	72,949	108 s.	GAL[11]
pr439.tsp	438	5	1	100	147,308	2%	2 s.	143,785	269.5 s.	GAL[11]
pr1002.tsp	**1001**	**5**	**1**	**220**	**329,128**	**-2%**	**22 s.**	**334,351**	**1524 s.**	**GAL[11]**
eil51.tsp	50	2	23	27	469	6%	29 ms.	442.32		CPLEX[8]
eil51.tsp	50	3	15	20	492	6%	14 ms.	464.11		CPLEX[8]
eil51.tsp	50	5	7	12	547	3%	23 ms.	529.70		CPLEX[8]
eil51.tsp	50	7	5	10	619	2%	16 ms.	605.21		CPLEX[8]
berlin52.tsp	51	2	10	41	8,283	7%	46 ms.	7,753.89		CPLEX[8]
berlin52.tsp	51	3	10	27	8,407	4%	81 ms.	8,106.85		CPLEX[8]
berlin52.tsp	51	5	6	17	9,577	5%	21 ms.	9,126.33		CPLEX[8]
berlin52.tsp	51	7	4	17	11,490	16%	72 ms.	9,870.02		CPLEX[8]
eil76.tsp	75	2	36	52	586	5%	23 ms.	558.59		CPLEX[8]
eil76.tsp	75	3	21	36	612	6%	51 ms.	579.30		CPLEX[8]
eil76.tsp	75	5	12	30	724	6%	27 ms.	680.67		CPLEX[8]
eil76.tsp	75	7	9	22	788	4%	37 ms.	759.90		CPLEX[8]
rat99.tsp	98	2	46	52	1,478	9%	52 ms.	1,350.73		CPLEX[8]
rat99.tsp	98	3	27	36	1,647	8%	51 ms.	1,519.49		CPLEX[8]
rat99.tsp	98	5	13	30	1,911	3%	122 ms.	1,855.83		CPLEX[8]
rat99.tsp	98	7	9	22	2,336	2%	126 ms.	2,291.82		CPLEX[8]

The table header contains 3 main columns: The name of an instance, the number of clients (n), the number of salesmen (k), and the upper and lower limits $(m_{\min} \textit{ and } m_{\max})$. Column *PCI*-algorithm shows the cost of the solutions obtained by our algorithm $(C(PCI))$, parameter $(Error_{PCI})$ specifies the approximation gap $Error_{PCI} = \left(\frac{C(BKS)-C(PCI)}{C(BKS)}\right)100\%$ of algorithm PCI compared to the cost of the Best Known Solutions $(C(BKS))$. Columns $(Time_{PCI})$ and cost $(C(BKS))$ specify the time and the cost, respectively, delivered by our algorithm. Column $(Time_{BKS})$ specifies time required by the algorithm with the Best Known Solution for a corresponding instance.

As it can be seen, *PCI*-algorithm obtained a lower cost solution for instance *pr1002.tsp*, and 16 instances with an approximation gap less than or equal to 7%. The execution time of *PCI*-algorithm is considerably lower than that of the state-of-the-art algorithms, in average, our algorithm is 132 times faster than two best-known state-of-the art algorithms [7,9–14] Hence, we hope that our algorithm has a potential to solve larger-sized instances.

4. Conclusions and Future Work

The proposed *PCI*-algorithm for the Bounded Single-Depot MTSP has obtained a best currently available solution for one of the publicly available benchmark instances and it provides an approximation gap not larger than 7% for 16 benchmark instances. Not less importantly, *PCI*-algorithm is too fast compared to the existing algorithms in the literature. This gives us a hope that it can give solutions for very large-scale instances.

Apparently, the partition phase 1 gives a good initial clustering of the set of vertices by making reasonable selection of vertices for each of the k subsets. At phase 2, the algorithm for TSP generates already good solutions which are further essentially improved at phase 3.

For future work, we plan to continue to develop the algorithm using new alternative procedures mainly at phase 3. We also plan to generate new benchmark instances for the Bounded Single-Depot MTSP. We also plan to convert TSPLIB benchmark [15] instances to the Bounded Single-Depot MTSP instances.

Author Contributions: Conceptualization, N.V.; Methodology, V.P-V.; Software and data recollection J.A.H.; Validation, N.V.; Formal Analysis, N.V.; Investigation, V.P-V.-N.V; Resources administrated by J.A.H.; Writing—original draft preparation, V.P-V; Writing—review and editing, N.V.; Visualization, V.P-V and N.V.; Supervision, N.V.; Project administration, N.V.; All authors have read and agreed to the published version of the manuscript

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bektas, T. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega* **2006**, *34(3)*, 209-219.
2. Kara, I.; & Bektas, T. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, **2006**, *174(3)*, 1449-1458.
3. Cheikhrouhou, O. & Khoufi, I. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. In *Computer Science Review* **2021**, *40*, 100369.
4. Vakhania, N.; Hernandez, J.A.; Alonso-Pecina, F.; Zavala, C. A Simple Heuristic for Basic Vehicle Routing Problem. *J. Comput. Sci.* **2016**, *3*, 39.
5. Pacheco-Valencia, V.; Hernández, J. A.; Sigarreta, J. M.; Vakhania, N. Simple Constructive, Insertion, and Improvement Heuristics Based on the Girding Polygon for the Euclidean Traveling Salesman Problem. *Algorithms* **2020**, *13(1)*, 5.
6. Jünger, M.; Reinelt, G.; Rinaldi, G. The traveling salesman problem. In *Handbooks in Operations Research and Management Science*; Elsevier Science B.V.; Sara Burgerhartstraat 25; P.O. Box 211, 1000 AE Amsterdam, The Netherlands, **1995**, *Volume 7*, pp. 225–330.
7. Junjie, P. & Dingwei, W. (2006, August). An ant colony optimization algorithm for multiple travelling salesman problem. In *First International Conference on Innovative Computing, Information and Control* **2006** *Volume I* (ICICIC'06) (*Vol. 1*, pp. 210-213). IEEE.
8. Necula, R.; Breaban, M.; & Raschip, M. Performance evaluation of ant colony systems for the single-depot multiple traveling salesman problem. *In International Conference on Hybrid Artificial Intelligence Systems.* Springer, Cham. **2015**, *(pp. 257-268)*.
9. Harrath, Y.; Salman, A. F.; Alqaddoumi, A.; Hasan, H.; & Radhi, A. A novel hybrid approach for solving the multiple traveling salesmen problem. In *Arab Journal of Basic and applied sciences* **2019**, *26(1)*, 103-112.
10. Hosseinabadi, A. A.; Kardgar, M.; Shojafar, M.; Shamshirband, S.; & Abraham, A. GELS-GA: hybrid metaheuristic algorithm for solving multiple travelling salesman problem. In *2014 14th International Conference on Intelligent Systems Design and Applications*, **2014**, (pp. 76-81). IEEE.
11. Lo, K. M.; Yi, W. Y.; Wong, P. K.; Leung, K. S.; Leung, Y.; & Mak, S. T. A genetic algorithm with new local operators for multiple traveling salesman problems. *International Journal of Computational Intelligence Systems*, **2018**, *11(1)*, 692-705.
12. Rostami, A. S.; Mohanna, F.; Keshavarz, H.; & Hosseinabadi, A. R. Solving multiple traveling salesman problem using the gravitational emulation local search algorithm. *Applied Mathematics & Information Sciences*, **2015**, *9(2)*, 1-11.
13. Yousefikhoshbakht, M.; & Sedighpour, M. A combination of sweep algorithm and elite ant colony optimization for solving the multiple traveling salesman problem. *Proceedings of the Romanian academy A* **2012**, *13(4)*, 295-302.
14. Yousefikhoshbakht, M.; Didehvar, F.; & Rahmati, F. (2013). Modification of the ant colony optimization for solving the multiple traveling salesman problem. *Romanian Journal of Information Science and Technology*, **2013**, *16(1)*, 65-80.
15. UNIVERSITÄT HEIDELBERG, INSTITUT FÜR INFORMATIK,GERHARD REINELT, *Symmetric traveling salesman problem (TSP).* consulted in: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, april 15th, 2021.
16. UAIC, Alexandru Ioan Cuza University of Iaşi, *Benchmark data for the Single-Depot Multiple Traveling Salesman Problem (multiple-TSP)* consulted in: https://profs.info.uaic.ro/~mtsplib/, jul 13th, 2021.
17. CPLEX *IBM CPLEX Optimizer* consulted in: https://www.ibm.com/analytics/cplex-optimizer, , aug 30th, 2021.