Computers & Industrial Engineering 90 (2015) 390–401

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A general variable neighborhood search heuristic for multiple traveling salesmen problem ☆

Banu Soylu

Department of Industrial Engineering, Erciyes University, 38039 Kayseri, Turkey

ARTICLE INFO

Article history:
Received 18 March 2015
Received in revised form 26 September 2015
Accepted 14 October 2015
Available online 10 November 2015

Keywords:
Multiple traveling salesmen problem
Minmax *m*TSP
Minsum *m*TSP
General variable neighborhood search

ABSTRACT

In this study, we consider the multiple traveling salesmen problem, which is the more general version of the single traveling salesman problem as it includes $m > 1$ salesmen starting and ending their tours at a fixed depot. We take into consideration two objective functions separately: one is to minimize the longest tour length and the other is to minimize the total length of all tours. A general variable neighborhood search, a well-known heuristic for combinatorial problems, is proposed for the *m*TSP. We test the performance of the heuristic on some test problems from the literature and compare it with existing approaches. We also apply the heuristic to a real life problem, which exists in the traffic signalization network of Kayseri province in Turkey, and obtain a considerable improvement.

1. Introduction

The multiple traveling salesmen problem (*m*TSP) is the more general version of the single traveling salesman problem. The *m*TSP arises when $m > 1$ disjoint tours, starting and ending at a fixed depot, need to be constructed and each vertex should be assigned once in a tour. The *m*TSP is strongly NP-hard since the TSP is the special case of the *m*TSP.

Although the most common objective function is to minimize the sum of the length (*minsum*) of all tours in the *m*TSP, minimizing the longest tour length (*minmax*) objective is also appropriate for real life problems since it leads to more balanced workload for each salesman assigned to each route. The *minmax* *m*TSP finds application in scheduling problems in addition to routing problems as well. It is closely related to the scheduling of jobs on the identical parallel machines problem by minimizing the makespan objective.

Different formulations and exact solution algorithms for the *minsum* *m*TSP have been presented in the literature (Bektas, 2006; Gavish, 1976; Gavish & Srikanth, 1986; Kara & Bektas, 2006; Laporte & Nobert, 1980; Sarin, Sherali, Judd, & Tsai, 2014; Svestka & Huckfeldt, 1973). A recent review and performance comparison of 32 formulations for the asymmetric *minsum* *m*TSP were conducted by Sarin et al. (2014). Although the exact solution procedures and optimum results have been extensively presented for the *minsum* *m*TSP in the literature, the study by França, Gendreau, Laporte, and Müller (1995) is the only one proposing exact algorithms and results for the *minmax* *m*TSP problem. Bertazzi, Golden, and Wang (2015) present bounds on the maximum tour length and total tour length of the *minsum* and *minmax* *m*TSP.

Since the problem size of the *m*TSP that can be solved exactly is very limited, different heuristic approaches have been developed in the literature. Russell (1977) presented one of the first heuristic approaches (*m*tour) for the *minsum* *m*TSP, which first constructs a single tour with a well-known Lin–Kernighan (1973) procedure, and then partitions the set of links into *m* subsets. Potvin, Lapalme, and Rousseau (1989) also presented a technique based on the partitioning of a single tour into a number of smaller tours with a variant of the *k*-opt exchange procedure. Frederickson, Hecht, and Kim (1978) proposed approximation algorithms, which are *k*-nearest insert, *k*-nearest neighbor and tour splitting, for the *minmax* *m*TSP. Na (2006) proposed a two-phase heuristic approach for the no-depot *minmax* *m*TSP where, in the first phase, *m* tours are constructed and in the second phase these tours are improved.

Apart from classic heuristic approaches, bio-inspired algorithms such as evolutionary and genetic algorithms have been presented for the *m*TSP in the literature.

Carter and Ragsdale (2006) developed a genetic algorithm, which uses a two-part chromosome, and evaluates its performance on *minsum* and *minmax* *m*TSP problems. Recently, Yuan, Skinner, Huang, and Liu (2013) proposed a new crossover operator for solving the *m*TSP with genetic algorithms. They evaluated the performance of the crossover operator for *minsum* and *minmax* *m*TSP instances. Singh and Baghel (2009) presented a new genetic representation, which is a set of *m* tours, and evaluated it over *minsum*

☆ This manuscript was processed by Area Editor Elghazali Talbi.

E-mail address: bsoylu@erciyes.edu.tr

http://dx.doi.org/10.1016/j.cie.2015.10.010
0360-8352/

and *minmax* *m*TSP instances. Brown, Ragsdale, and Carter (2007) designed a grouping genetic algorithm. Kiraly and Abonyi (2011) proposed a genetic algorithm with multi-chromosome representation for the *minsum* *m*TSP. Tang, Liu, Rong, and Yang (2000) proposed a modified genetic algorithm for the hot rolling scheduling problem which is modeled as a *minsum* *m*TSP. Yu, Wang, Lin, Li, and Wu (2012) proposed a hierarchical approach where the top level of the hierarchy uses a genetic algorithm while the bottom level uses a branch-and-cut and Lin–Kernighan algorithms. Larki and Yousefikhoshbakht (2014) presented an evolutionary algorithm which integrates the modified imperialist competitive algorithm and Lin–Kernighan algorithm in order to solve the *minsum* *m*TSP. Sedighpour, Yousefikhoshbakht, and Mahmoodi Darani (2012) proposed a two stage genetic algorithm, where at the first stage a modified genetic algorithm solves the *minsum* *m*TSP and at the second stage the 2-opt operator is applied to improve the solution.

Swarm intelligence based approaches, such as artificial bee colony, ant colony optimization, and invasive weed optimization have also been applied to the *m*TSP successfully. Recently, Venkatesh and Singh (2015) proposed two approaches, which are based on artificial bee colony and invasive weed optimization, for the *minsum* and *minmax* *m*TSP. Liu, Li, Zhao, and Zheng (2009) proposed an ant colony algorithm for the *minsum* and *minmax* *m*TSP. Yousefikhoshbakht, Didehvar, and Rahmati (2013) hybridized the ant colony algorithm with swap, insert and 2-opt approaches for the *minsum* *m*TSP. Another ant colony algorithm was presented by Ghafurian and Javadian (2011) for the multi-depot variant of the *m*TSP.

One of the first applications of neural networks to the *m*TSP is due to Wacholder, Han, and Mann (1989). They developed a neural network algorithm for the *minsum* *m*TSP. Somhom, Modares, and Enkawa (1999) introduced the competition based neural network for the *minmax* *m*TSP. They presented a competition rule to determine the winner city to be added into the current tour. We also adapted this competition rule and its different variants into our initialization heuristic. Modares, Somhom, and Enkawa (1999) proposed a neural network based algorithm for the *minmax* *m*TSP. Hsu, Tsai, and Chen (1991) presented a neural network approach, which is based on the self-organized feature map model, for the *minsum* *m*TSP.

Other metaheuristic approaches applied to *m*TSP are the simulated annealing and the tabu search. Song, Lee, and Lee (2003) presented a simulated annealing algorithm for the *minsum* *m*TSP. Golden, Laporte, and Taillard (1997) proposed the adaptive memory tabu seach algorithm for the *minmax* *m*TSP.

In this study, we propose a general variable neighborhood search (GVNS) approach for the *m*TSP. Our motivation in dealing with this problem is to apply it to the traffic signalization network in a city of Turkey. This network currently includes 170 junctions and is still growing with new investments. It is divided into six tours currently, and a team visits a tour per day routinely. However, as the tour lengths are not balanced, the team works over schedule on some days and under schedule on other days. This problem is typically *minmax* *m*TSP. Since the network is huge not to allow finding of an optimum solution, we decided to develop a heuristic approach especially for the *minmax* *m*TSP. The variable neighborhood search (VNS) algorithm (Hansen & Mladenović, 2001) has been successfully applied to many combinatorial optimization problems including several vehicle routing problems (see for instance (Imran, Salhi, & Wassan, 2009; Mladenović, Urošević, & Hanafi, 2013; Mladenović, Urošević, & Ilić, 2012; Polat, Kalayci, Kulak, & Günther, 2015; Zhao, Chen, & Li, 2012)). As can be seen from the literature review above, although metaheuristics rather than classical heuristic approaches have been receiving great deal of attention for solving the *m*TSP, the VNS is rarely applied to the *m*TSP (according to our best knowledge, there is only one study by (Zhao et al., 2012) for the *m*TSP with deadlines). The VNS systematically searches neighbors of a solution and terminates with a local optimum. In GVNS, the search is applied by using different neighborhoods (Mladenović et al., 2013; Mladenović et al., 2012; Hansen, Mladenović, & Pérez, 2008; Hansen, Mladenović, & Pérez, 2010). The performance of the proposed GVNS heuristic is tested by using *m*TSP instances and also compared with the optimum results and those of some approaches from the literature. Finally, it is applied to a traffic signalization network in the Kayseri province of Turkey.

In Section 2, we define the *m*TSP problem and give the formulations. In Section 3, we develop the GNVS algorithm. In Section 3, we provide the computational test results. In Section 4, we introduce the traffic signalization network application and present results. In Section 5, we give our concluding remarks and further research directions.

2. Problem definition

Let $G = (V, A)$ be a complete graph, where V is the set of n vertices (nodes) including a depot and A is the set of arcs (edges). Each arc $(i,j) \in A$ has a non-negative length of d_{ij}. If there are m salesmen, the aim is to determine a tour for each salesman such that it starts and ends at the depot and at least one node is visited, and all nodes are visited once by any salesman. The objective function can be either *minsum* or *minmax*.

The assignment based formulation of the *minsum* *m*TSP with Miller–Tucker–Zemlin (MTZ) (Miller, Tucker, & Zemlin, 1960) subtour elimination constraints is given as follows (Bektas, 2006; Gavish, 1976; Kara & Bektas, 2006; Sarin et al., 2014; Svestka & Huckfeldt, 1973):

$$x_{ij} = \begin{cases} 1 & \text{if node } i \text{ precedes node } j \text{ on a tour} \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j = 1,2,\ldots,n, \;\; i \neq j$$

$$u_i = \textit{visiting rank of city } i \textit{ in order} \quad \forall i = 2,3,\ldots,n, \quad u_1 = 0$$

$$\textit{Minsum mTSP}: \quad \textit{Minimize } Z = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} d_{ij} x_{ij} \tag{1}$$

Subject to

$$\sum_{j=2}^{n} x_{1j} = m \tag{2}$$

$$\sum_{i=2}^{n} x_{i1} = m \tag{3}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall j = 2,\ldots,n \tag{4}$$

$$\sum_{\substack{j=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall i = 2,\ldots,n \tag{5}$$

$$u_i - u_j + p x_{ij} \leqslant p - 1 \quad \forall 2 \leqslant i \neq j \leqslant n \tag{6}$$

$$1 \leqslant u_i \leqslant p \quad \forall i = 2,\ldots,n \tag{7}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j = 1,2,\ldots,n, \; i \neq j \tag{8}$$

Here we assume that node 1 is the depot and the visiting rank of node 1, u_1, is 0. Constraints (2) and (3) ensure that exactly m salesmen depart from and return to the depot. Constraint sets (4) and (5) are the assignment constraints requiring that each node (except the

depot) should be preceded by and precedes exactly one another node. Constraint sets (6) and (7) are the MTZ subtour elimination constraints where $2 \leqslant p \leqslant n - m$ denotes the maximum number of cities that can be visited by any salesman.

Svestka and Huckfeldt (1973) and Bellmore and Hong (1974) also proposed the equivalent TSP formulation for the *m*TSP. According to the results of a comparison study by Sarin et al. (2014), which evaluates 32 different formulations for the *minsum* *m*TSP, the above formulation requires the smallest CPU time to obtain the integer optimal solution. We modified it by adding the cumulative distance variable, T_j, and obtained the *minmax* *m*TSP formulation as follows:

$T_j =$ partial tour length from the depot to node $j \quad \forall j = 1, 2, \ldots, n$

In this case, T_1 is the maximum tour length from depot to depot. Note that if node i precedes node j, then $T_i < T_j$.

$$\textit{Minmax mTSP}: \quad \textit{Minimize } T_1 \tag{1'}$$

Subject to

(2)–(8)

$$T_j \geqslant T_i + d_{ij} + M(x_{ij} - 1) \quad \forall i = 2, 3, \ldots, n,\ j = 1, 2, \ldots, n \tag{9}$$

$$T_j \geqslant d_{1j} x_{1j} \quad \forall j = 2, 3, \ldots, n \tag{10}$$

$$T_j \geqslant 0 \quad \forall j = 1, 2, \ldots, n \tag{11}$$

Since T_1 would be greater than all tour lengths, minimizing T_1 as in (1′) would also minimize the maximum tour length. Constraint set (9) ensures that the length of the partial tour from depot to node j should be greater than the partial tour length to node i and the distance between nodes i and j, if node i precedes node j where M is a big positive number. Otherwise, it is redundant. Constraint set (9) by itself does not consider the distance from depot to the first destination on a tour since node 1 is labeled as the final arrival destination (then the first T_i, $i \neq 1$ in the order would be 0). So, the constraint set (10) requires that the length of the partial tour from depot to the first destination j on a tour should be greater than the distance between the depot and the first destination j.

Although the medium size ($n = 30, 40$ and $m = 2, 3, 4$) instances of the *minsum* *m*TSP can be solved optimally within a reasonable time limit (CPU time < 3600 s), only small instances of *minmax* *m*TSP can be solved optimally within the same time limit. In Table 1, we performed a preliminary comparison by using instances and results for the *minsum* *m*TSP from the literature. Sarin et al. (2014) provide *minsum* *m*TSP results for $m = 2$, 3 and 4 salesmen over Br17 and Ftv35 data sets (Reinelt, 1991). The *opt* column presents whether the optimum integer solution is found within 3600 s or not, while the CPU column shows the solution time of Cplex 12.1 solver (Cplex, 2010) for the corresponding model and the *#ofBBnodes* column presents the BB nodes generated by the branch-and-cut algorithm. In addition to Br17 and Ftv35 data sets, we also generated Br17_10 and Br17_13 data sets which include the first 10 and 13 nodes of Br17, respectively. Since we are able to solve only small instances with the *minmax* *m*TSP, we also derive 10, 15 and 20 nodes instances from our asymmetric 170 nodes application data set. The huge difference between the solution times of both models can be observed in Table 1. Although the optimal solution of the *minsum* *m*TSP can be found for all instances in less than 15 s, the optimal solution of the *minmax* *m*TSP can only be found for $n = 10$ instances within 3600 s. The large number of BBnodes generated during the solution of the *minmax* *m*TSP also indicates the difficulty of this problem over the *minsum* *m*TSP.

Table 1
Some performance results for *minsum* and *minmax* *m*TSP.

Data set	n	m	*Minsum* *m*TSP			*Minmax* *m*TSP		
			opt	CPU (s)	*#ofBBnodes*	*opt*	CPU (s)	*#ofBBnodes*
Br17	17	2	√	0.80*	2577*	–	3600	254,256
		3	√	0.72*	2134*	–	3600	3,774,520
		4	√	0.41*	273*	–	3600	1,944,836
	10	2	√	0.11	7	√	21.68	80,000
		3	√	0.09	6	√	5.54	44,742
		4	√	0.08	9	√	1.51	6037
	13	2	√	0.16	72	–	3600	1,109,624
		3	√	0.11	52	–	3600	605,128
		4	√	0.11	31	–	3600	5,728,277
Ftv33	34	2	√	1.09*	22*	–	3600	261,997
		3	√	0.95*	281*	–	3600	132,936
		4	√	1.36*	120*	–	3600	114,584
Ftv35	36	2	√	1.13*	1008*	–	3600	240,369
		3	√	1.02*	422*	–	3600	164,270
		4	√	0.80*	357*	–	3600	145,978
TrN170	10	2	√	0.09	104	√	265.73	1,435,805
		3	√	0.08	99	√	108.05	654,371
		4	√	0.08	34	√	26.21	146,752
	15	2	√	0.44	565	–	3600	2,112,141
		3	√	0.34	189	–	3600	3,728,698
		4	√	0.22	206	–	3600	3,721,691
	20	2	√	2.67	4402	–	3600	508,050
		3	√	14.06	18,615	–	3600	452,575
		4	√	7.32	13,751	–	3600	542,691

* Results from Sarin et al. (2014).

3. The general variable neighborhood search algorithm

The GVNS is a variant of the VNS, where different neighborhoods are used. Let $\boldsymbol{x}$ be a feasible solution and the set $N_k(\boldsymbol{x})$ be the set of all solutions in the kth neighbor of the solution $\boldsymbol{x}$. In this case, each element $\boldsymbol{x}' \in N_k(\boldsymbol{x})$ is k away from the solution $\boldsymbol{x}$. Here, we measure the distance between solutions $\boldsymbol{x}$ and $\boldsymbol{x}'$ by counting the nodes transferred to another tour. This type of local search is called a variable neighborhood descent (VND) (Mladenović et al., 2012). In sequential VND (*Seq*_VND), the neighbors are visited according to an order. The components of the GVNS algorithm include an initial feasible solution, a neighborhood search structure, a terminating condition and a shaking procedure. For different variants of the GVNS algorithm and their successful applications see recent surveys by Hansen et al. (2008) and Hansen et al. (2010).

The pseudo-code of the GVNS is given in Algorithm 1. Here the parameter l_{max} is the maximum number of different neighborhood structures in *Seq*_VND, t_{max} is the maximum running time allowed, k_{max} is the maximum number of neighborhoods in the shaking stage, and $\boldsymbol{d}$ is an $n \times n$ distance matrix. After an initial solution is found, the local search starts. The *Seq*_VND procedure searches all neighbors of a given solution $\boldsymbol{x}'$ according to an order and returns the best found solution $\bar{\boldsymbol{x}}$. Then it is immediately compared with the incumbent solution $\boldsymbol{x}$, and if it is better than the incumbent, the incumbent is replaced. Otherwise, a shaking procedure is applied to shift another solution. As the solution $\bar{\boldsymbol{x}}$ does not improve, the magnitude of shaking is increased to allow escaping from the local optima. The algorithm ends whenever a time limit is reached.

Algorithm 1. The steps of the GVNS algorithm

Procedure *GVNS*($\boldsymbol{d}, l_{max}, k_{max}, t_{max}$)
1: $\boldsymbol{x} \leftarrow$ *Initialization*($\boldsymbol{d}$) ! get a feasible solution
2: *While*($t < t_{max}$) do
3: $k \leftarrow 1$
3: *While* ($k < k_{max}$) do
4: $\boldsymbol{x}' \leftarrow$ *Shake*($\boldsymbol{x}, k$)
5: $\bar{\boldsymbol{x}} \leftarrow$ *Seq_VND*($\boldsymbol{x}', l_{max}$)
6: *if* $f(\bar{\boldsymbol{x}}) < f(\boldsymbol{x})$ *then*
7: $\boldsymbol{x} \leftarrow \bar{\boldsymbol{x}}$
8: $k \leftarrow 1$
9: *else* $k \leftarrow k + 1$
10: *endif*
11: *endwhile*
12: *endwhile*

3.1. Initialization

A competition rule based heuristic approach is used to get an initial feasible solution for the *minmax* *m*TSP. This competition rule was initially proposed by Somhom et al. (1999) and used in an adaptive neural network algorithm. In this study, we apply it to construct m feasible tours from scratch and each time define a winner tour through a competition based approach. For this purpose, first, the nodes are sorted in a non-decreasing order of distance to the depot. Then, m small tours, which start and end at the depot, are initially defined by the first m nodes in the order. The tours are enlarged by selecting a winner tour, v^*, for each unassigned node i. Then node i is assigned after the last node j^- of tour v according to the following rule (Somhom et al., 1999):

$$v^* = \underset{v}{\operatorname{argmin}} \left\{ d_{v(j^-)i} * \left[1 + \frac{length_v - avg.lenth}{avg.length} \right] \right\} \quad \forall v = 1, 2, \ldots, m \tag{12}$$

where $d_{v(j^-)i}$ is the distance between an unassigned node i and the last assigned node j^- of tour v. The $length_v$ is the total length of tour v, while the $avg \cdot lenth$ is the average tour length currently. Here the second part in the multiplication aims to increase or decrease the chance of selecting node i into tour v in such a way that if $length_v < avg \cdot lenth$, the chance increases otherwise it decreases. This part also prevents deviation from average tour length, which is the objective of the *minmax* *m*TSP.

Whenever all nodes are assigned, a feasible solution is obtained. The pseudo-code of the heuristic is given in Algorithm 2.

Algorithm 2. The steps of competition rule based heuristic

Procedure *Initialization*($\boldsymbol{d}$)
1: *Sort* $d_{[1]1} < d_{[2]1} < \cdots < d_{[n-1]1}$! Sort nodes in non-decreasing order of d_{i1}, $\forall i$
2: *For* v:1 *to* m *do* ! obtain m small tours
3: $tour_v \leftarrow tour_v \cup node_{[v]}$
4: *endfor*
5: *For* $i : m + 1$ *to* n *do*
6: $v^* = \operatorname{argmin}_v \left\{ d_{v(j^-)i} * \left[1 + \frac{length_v - avg.lenth}{avg.length} \right] \right\}$ $\forall v = 1, 2, \ldots, m$
7: $tour_{v*} \leftarrow tour_{v*} \cup i$
8: *endfor*
9: *For* v:**1** *to* m *do*
10: if node i precedes node j in tour v
11: $x_{ij} = 1$
12: else $x_{ij} = 0$
13: *endfor*
13: Return x

To adapt this procedure to the *minsum* *m*TSP, the winner tour for node i is determined as the one which increases the tour length least. For this purpose, the arc between node j^- and node 1 is broken and node i is inserted. This rule is given in Eq. (13).

$$v^* = \underset{v}{argmin} \left\{ d_{v(j^-)i} + d_{i1} - d_{v(j^-)1} \right\} \quad \forall v = 1, 2, \ldots, m \tag{13}$$

3.2. Shaking

In the shaking procedure, the incumbent solution $\boldsymbol{x}$ is perturbed to obtain a random solution $\boldsymbol{x}'$ in the kth neighborhood of $\boldsymbol{x}$. It is useful to escape from the local optima. Depending on the k value, this random solution belongs to the close or far neighbors of $\boldsymbol{x}$, and it can at most be k_{max} away from the solution $\boldsymbol{x}$. For this purpose, a perturbation scheme developed by Salhi and Rand (1987) is applied. It takes a random node from a randomly chosen route and relocates that node into a randomly chosen position of another randomly chosen route. The pseudo-code of the shaking procedure is given in Algorithm 3.

Algorithm 3. The steps of the shaking procedure

Procedure *Shake*($\boldsymbol{x}, k$)
1: *While* $k > 0$ *do*
2: $v \leftarrow Rand[1, m]$! Choose a random tour
3: $i \leftarrow Rand[1, |tour_v|]$! Chose a random node in $tour_v$
4: $t \leftarrow Rand[1, m]$ *and* $t \neq v$! Choose another random tour
5: $j \leftarrow Rand[1, |tour_t|]$! Chose a random node in $tour_t$
6: *Insert node* i *into* $tour_t$ *between nodes* j *and* $j + 1$
$x_{i-1i} = 0,\ x_{ii+1} = 0,\ x_{i-1i+1} = 1$! Remove node i
$x_{jj+1} = 0,\ x_{ji} = 1,\ x_{ij+1} = 1$! Insert node i
7: $k \leftarrow k - 1$
8: *endwhile*
9: Return x

3.3. Neighborhood search structures

The most important part of GVNS is to use appropriate local search schemes. Neighborhood structures and the order of neighborhood exploration schemes affect the performance of the Sequential VND. Five neighborhood structures (i.e. $l_{max} = 5$), which are *one-point move*, two types of *or-opt move*, *two-point move* and *three-point move*, are used in this study. These structures were initially developed for the TSP and vehicle routing problems (see (Cordeau, Laporte, Savelsbergh, & Vigo, 2006; Groër, Golden, & Wasil, 2010; Kindervater & Savelsbergh, 1997)), and are modified to the *m*TSP in this study. These operators are explained below and illustrated in Fig. 1. The computational complexity of these local search operators is $O(n^2)$ for a given solution $\boldsymbol{x}'$.

3.3.1. One-point move

This operator searches for 1-nighbors of a given solution $\boldsymbol{x}'$. It basically relocates a selected node into a new position in another

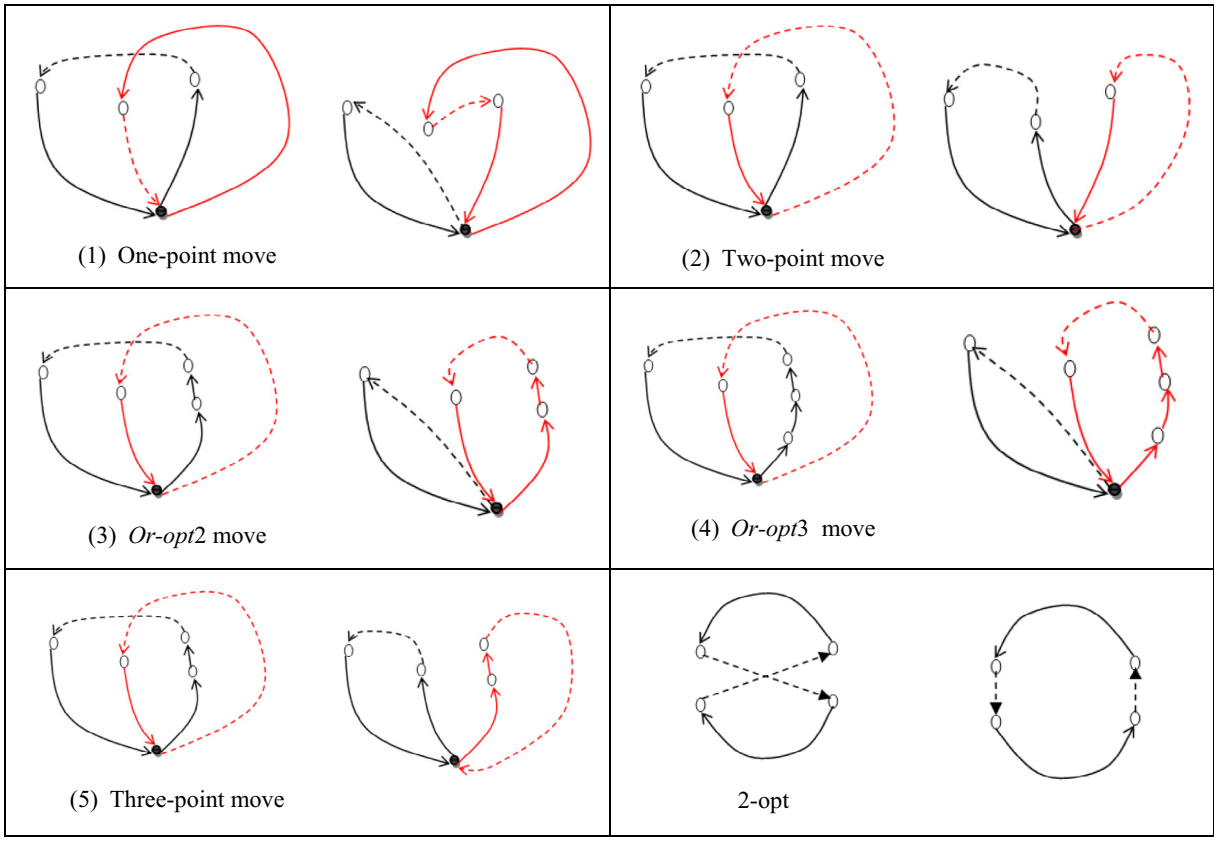


Fig. 1. Local search operators used.

tour. This selected node belongs to a randomly selected tour (in the *minsum* *m*TSP) and the tour having the maximum length (in the *minmax* *m*TSP). This operation is repeated for all nodes of the selected tour and for all available positions of other tours. Whenever a one-point move is realized, a new solution $\boldsymbol{x}''$ is obtained. To perform another one-point move, the algorithm returns back to the current solution $\boldsymbol{x}'$. After performing all possible one-point moves from the solution $\boldsymbol{x}'$, the best solution $\boldsymbol{x}'' \in N_1(\boldsymbol{x}')$, i.e. a solution in the 1-neighborhood of $\boldsymbol{x}'$, is returned by the operator. The computational complexity of this local search operator is $O(n^2)$ for a given solution $\boldsymbol{x}'$ (since in the worst case $\frac{n}{2}$ nodes of the selected tour should be relocated into $\frac{n}{2}$ available positions).

3.3.2. Or-opt move

This type of move inserts a string of nodes into a new position. The size of this string is defined as two and three adjacent nodes in this study. These operators are called *or-opt*2 and *or-opt*3, respectively. This selected string belongs to a randomly selected tour (in *minsum* *m*TSP) and the tour having the maximum length (in *minmax* *m*TSP). This operation is repeated for all strings of the selected tour and for all available positions of other tours. After performing all possible or-opt moves from the solution $\boldsymbol{x}'$, the best solution $\boldsymbol{x}''$ is returned by the operator. The *or-opt*2 operator generates $\boldsymbol{x}'' \in N_2(\boldsymbol{x}')$ while *or-opt*3 operator generates $\boldsymbol{x}'' \in N_3(\boldsymbol{x}')$.

3.3.3. Two-point move

This move basically swaps the positions of two nodes. The selected node, belonging to a randomly selected tour (in *minsum* *m*TSP) and the tour having the maximum length (in *minmax* *m*TSP), is swapped with a node belonging to another tour. After performing all possible two-point moves from the solution $\boldsymbol{x}'$, the best solution $\boldsymbol{x}''$ is returned by the operator. This operator generates $\boldsymbol{x}'' \in N_2(\boldsymbol{x}')$.

3.3.4. Three-point move

As different from two-point move, a pair of adjacent nodes are selected and swapped with a node belonging to another tour. The selected pair of nodes belongs to a randomly selected tour (in the *minsum* *m*TSP) and the tour having the maximum length (in the *minmax* *m*TSP). After performing all possible three-point moves from the solution $\boldsymbol{x}'$, the best solution $\boldsymbol{x}''$ is returned by the operator. This operator generates $\boldsymbol{x}'' \in N_3(\boldsymbol{x}')$.

3.3.5. 2-opt move

This operator was initially developed by Croes (1958) for TSP and is used to improve the resulting tours (as an intra-tour improvement operator) in this study. It removes two non-adjacent arcs from a given tour and adds two new arcs, while maintaining the tour structure.

3.4. Sequential VND

In sequential VND, the neighbors of a given solution $\boldsymbol{x}'$ are visited according to a particular sequence. We determined this sequence as the non-decreasing order of neighborhood size. We also performed an experiment in the Computational Results Section to decide on the best sequence. Accordingly, the sequential VND is applied as given in Algorithm 4. If the best solution $\boldsymbol{x}''$ returned by the local search operator is better than the current best $\boldsymbol{x}'$, the 2_*opt* operator is applied for a possible intra-tour improvement and the resulting solution is replaced with the current best. Then the search is restarted with the smallest neighborhood local

search operator. Otherwise, the search is continued with a larger neighborhood search operator.

Algorithm 4. The steps of sequential VND

Procedure *Seq_VND*($\boldsymbol{x}', l_{max}$)
1: $l \leftarrow 1$
2: *While* $l \leqslant l_{max}$ *do*
3: *if* $l = 1$ do
4: $\boldsymbol{x}'' \leftarrow onepoint(\boldsymbol{x}')$
5: *endif*
6: *if* $l = 2$ do
7: $\boldsymbol{x}'' \leftarrow or\_opt2(\boldsymbol{x}')$
8: *endif*
9: *if* $l = 3$ do
10: $\boldsymbol{x}'' \leftarrow twopoint(\boldsymbol{x}')$
11: *endif*
12: *if* $l = 4$ do
13: $\boldsymbol{x}'' \leftarrow or\_opt3(\boldsymbol{x}')$
14: *endif*
15: *if* $l = 5$ do
16: $\boldsymbol{x}'' \leftarrow threepoint(\boldsymbol{x}')$
17: *endif*
18: *if* $f(\boldsymbol{x}'') < f(\boldsymbol{x}')$ *then*
19: $\hat{\boldsymbol{x}} \leftarrow 2\_opt(\boldsymbol{x}'')$
20: $\boldsymbol{x}' \leftarrow \hat{\boldsymbol{x}}$
21: $l \leftarrow 1$
22: *else* $l \leftarrow l + 1$
23: *endif*
24: *endwhile*
25: Return $\boldsymbol{x}'$

4. Computational results

The GVNS algorithm was coded in C++ and run on a 2.4 GHz workstation with 4 GB of RAM memory. The algorithms were tested on instances of *m*TSP test problems (Yuan et al., 2013; Carter & Ragsdale, 2006) of different size and the data set belongs to the 170 nodes traffic network in the Kayseri province of Turkey. In the first group of test problems (Yuan et al., 2013), there are three test problems with $n = 128$ cities and $m = 10$, 15 and 30 salesmen, and five small sized *m*TSP test problems called 11a, 11b, 12a, 12b, and 16. Test problems 11a, 12a and 16 comprise the first 11, 12 and 16 cities of the $n = 51$ test problem of (Carter & Ragsdale, 2006) respectively, whereas 11b and 12b test problems are derived from sp11 and uk12 data sets (visit http://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html for the first group of test problems). The second group of test problems (Carter & Ragsdale, 2006) includes twelve problems with $n = 51$, 100 and 150 cities and $m = 3$, 5, 10, 20 and 30 salesmen. This group is called the *m*TSP*n*. Our traffic network is presented in the Application Section. Ten replications (using a different random seed in the shaking phase) were performed for each problem. The average objective value (*Avg.*), standard deviation (SD) and the best found objective value (*best*) statistics are reported.

For all experiments, the running time t_{max} is set as n (# of nodes in the network) seconds as a result of the convergence analysis of the algorithm. Since we have five local search operators, l_{max} is naturally set as 5. The different levels of parameter k_{max} are also analyzed in the experimentation. Although the proposed GVNS was specially developed for *minmax* *m*TSP problems, we also performed experimentation on *minsum* *m*TSP problems.

We performed a preliminary analysis to determine the efficiency of our local search operators. For this purpose, we run the GVNS algorithm with each operator one by one on the *minmax* 3TSP150 problem. Here close results to the *all operators case* imply the strong affect of the operator. Additionally, we analyzed the performance by removing each operator one by one. In this case, high deviations from the *all operators case* imply the strong influence of the operator. This means that the results are much affected by the nonexistence of the considered operator. The results of ten replications are given in Table 2. Accordingly, the result of running the GVNS algorithm only with operator (1) or (3) is close to the result of the *all operators case*. Running with the operator (1) or (3) gives the best performance while running with the operator (4) gives the worst performance. We also observe the effectiveness of operators (1) or (3) when we exclude them. For instance, the performance declines considerably when the GVNS runs with all operators except (1). However, the performance declines less when the GVNS runs with all operators except (4). Since the existence or

Table 2
Evaluation of local search operators (for *minmax* 3TSP150).

Active operators	Avg.	SD	Best
Only (1)	13846.4	167.9	13664.9
Only (2)	14450.2	414.7	14056.3
Only (3)	13888.1	283.1	13474.3
Only (4)	15103.3	330.1	14545.3
Only (5)	14254.9	278.3	13732.9
All operators	**13627.8**	**189.5**	**13376.4**
Except (1)	14020.2	183.2	13676.7
Except (2)	13697.7	174.2	13363.2
Except (3)	13836.6	220.9	13522.4
Except (4)	13650.5	152.9	13436.6
Except (5)	13833.9	315.9	13519.5

Table 3
Results with different order of local search operators (*minmax* 3TSP150).

Order	Avg.	SD	Best
(1)-(2)-(3)-(4)-(5)	**13627.8**	189.5	**13376.4**
(2)-(1)-(3)-(4)-(5)	14008.8	225.5	13740.3
(1)-(2)-(4)-(3)-(5)	13641.4	141.4	13436.6
(2)-(1)-(4)-(3)-(5)	14032.4	224.6	13770.3
(1)-(2)-(3)-(5)-(4)	13647.6	155.7	13436.6
(2)-(1)-(3)-(5)-(4)	14032.4	224.6	13770.3

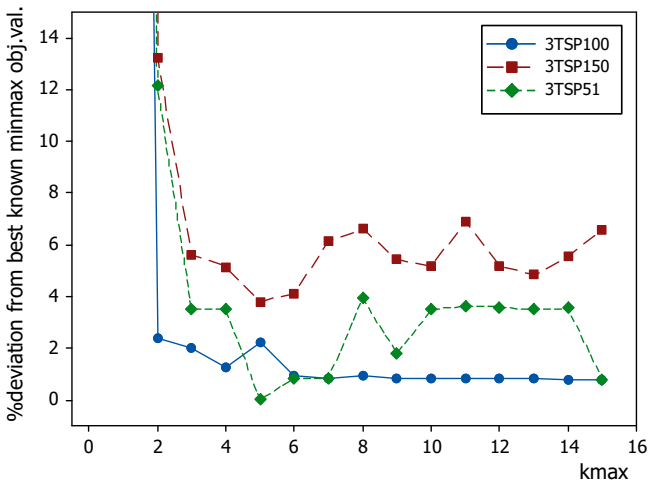


Fig. 2. Results with different k_{max} values.

nonexistence of each operator more or less affects the performance, we decided to include all operators in our experimentation.

Since the order of local search operators may affect the performance of the algorithm, we performed an analysis for the purpose of deciding the implementation order of local search operators. For this purpose, we grouped one-point and two-point move operators and changed the order within the group. We also grouped the remaining three operators and changed their orders within the group as well. Table 3 presents the results on the 3TSP150 problem over ten replications. Accordingly, the order (1)-(2)-(3)-(4)-(5) is better in terms of average and best performance. Hence, all experiments are performed by using this order.

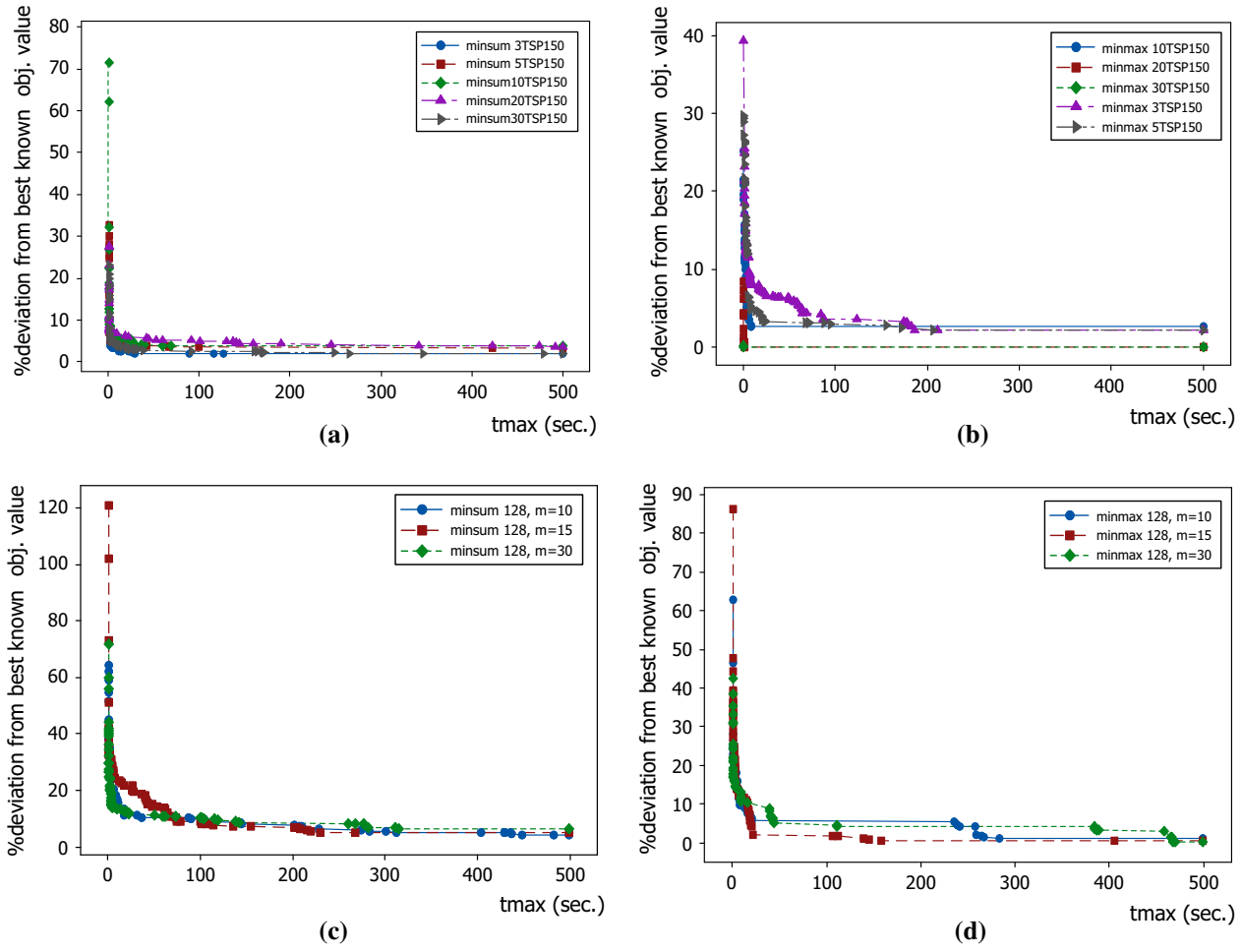


Fig. 3. Convergence results of the GVNS Seq-VND algorithm.

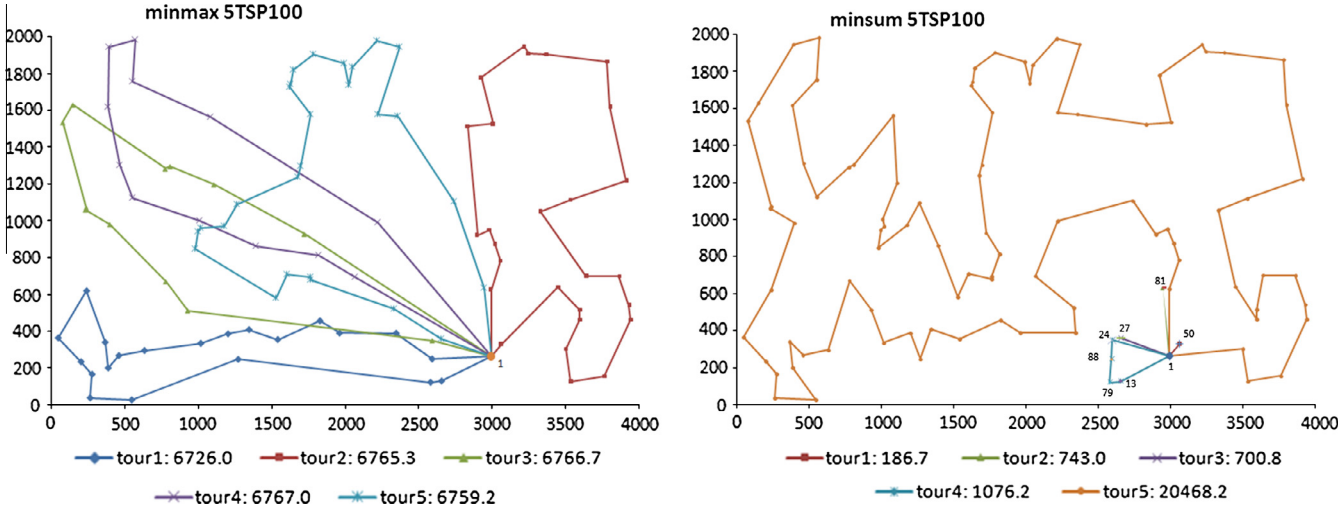


Fig. 4. Resulting tours with *minmax* and *minsum* objectives for a 5TSP100 instance.

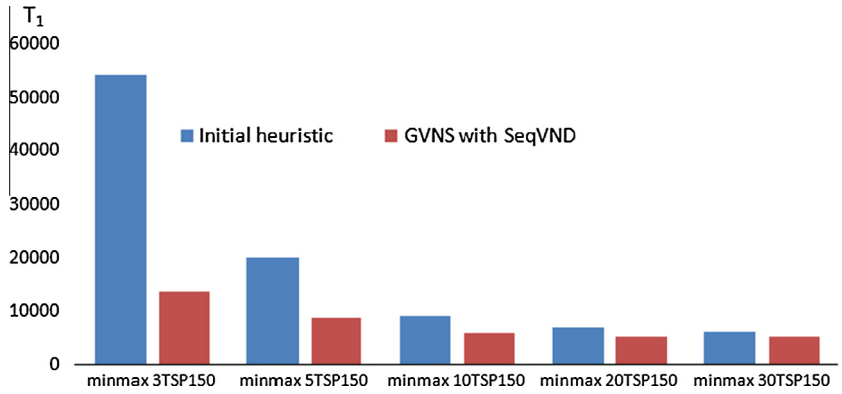


Fig. 5. Performance comparison of the initial heuristic and the GVNS Seq-VND algorithm for the *minmax* *m*TSP150.

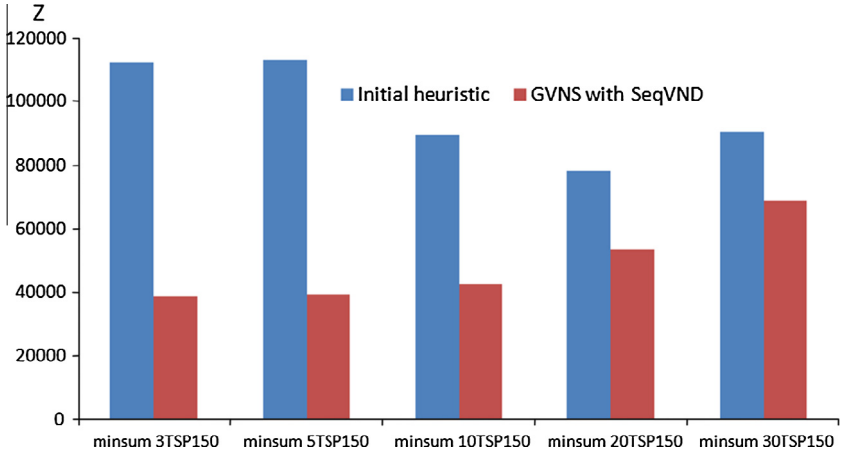


Fig. 6. Performance comparison of the initial heuristic and the GVNS Seq-VND algorithm for the *minsum* *m*TSP150.

Table 4
Performance results of the GVNS algorithm and comparison with other algorithms from the literature on the first group of test problems (Yuan et al., 2013).

Problem	*m*	Optimal	TCX (Yuan et al., 2013)		ABC(FC) (Venkatesh & Singh, 2015)		ABC(VC) (Venkatesh & Singh, 2015)		IWO (Venkatesh & Singh, 2015)		GVNS Seq-VND			GVNS Seq-VND2			
			Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	SD	Best	Avg.	SD	Best	% Imp.
Minmax																	
11a	3	77	**77**	**77**	**77**	**77**	**77**	**77**	**77**	**77**	**77**	0	**77**	NA			
11b	3	73	**73**	**73**	**73**	**73**	**73**	**73**	**73**	**73**	**73**	0	**73**	NA			
12a	3	77	**77**	**77**	**77**	**77**	**77**	**77**	**77**	**77**	**77**	0	**77**	NA			
12b	3	983	987	**983**	**983**	**983**	**983**	**983**	**983**	**983**	**983**	0	**983**	NA			
16	3	94	101	**94**	**94**	**94**	**94**	**94**	**94**	**94**	**94**	0	**94**	NA			
128	10	–	6716	5912	5243	4872	5058	4660	4914	4450	**3049**	66	**2980**	NA			
	15	–	5979	5295	4481	3819	4443	3958	4196	3665	**2411**	56	**2305**	NA			
	30	–	4814	4003	4025	3456	4127	3811	3725	3494	**2049**	45	**1980**	NA			
Minsum																	
11a	3	198	**198**	**198**	**198**	**198**	**198**	**198**	**198**	**198**	**198**	0	**198**	**198**	0	**198**	0
11b	3	135	**135**	**135**	**135**	**135**	**135**	**135**	**135**	**135**	**135**	0	**135**	**135**	0	**135**	0
12a	3	199	**199**	**199**	**199**	**199**	**199**	**199**	**199**	**199**	**199**	0	**199**	**199**	0	**199**	0
12b	3	2295	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	0	**2295**	**2295**	0	**2295**	0
16	3	242	247	**242**	**242**	**242**	**242**	**242**	243	**242**	**242**	0	**242**	**242**	0	**242**	0
128	10	–	42,335	39,478	33,072	30,799	29,208	26,482	25,886	24,514	25,532	561	24,749	**23,081**	342	**22,647**	10
	15	–	44,294	40,843	36,338	32,777	31,752	28,405	28,848	26,368	28,949	810	27,257	**25,766**	422	**25,204**	11
	30	–	54,696	50,809	47,533	43,599	44,212	41,754	40,875	39,579	42,560	548	41,737	**38,178**	330	**37,383**	10

NA: not applicable.

Fig. 2 presents the performance results of the GVNS algorithm on the 3TSP51, 100 and 150 test problems for different settings of k_{max} value. The horizontal axis shows the different levels of the parameter k_{max} while the vertical axis shows the percent deviation from the best known obj. value, i.e. $100 * \frac{(Z - Z^{best})}{Z^{best}}$. According to this figure, the best results are obtained around $k_{max} = 5$ on nearly all problems. Therefore, the parameter k_{max} is set to 5 for all instances.

Table 5
Performance results of the GVNS algorithm and comparison with other algorithms from the literature on the second group of test problems (Carter & Ragsdale, 2006).

Data set	m	GA2PC (Carter & Ragsdale, 2006)	GGA-SS (Singh & Baghel, 2009)	ACO (Liu et al., 2009)	TCX (Yuan et al., 2013)	ABC(FC) (Venkatesh & Singh, 2015)		ABC(VC) (Venkatesh & Singh, 2015)		IWO (Venkatesh & Singh, 2015)		GVNS Seq-VND			GVNS Seq-VND2			
		Avg.	Avg.	Avg.	Avg.	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	SD	Best	Avg.	SD	Best	% Imp.
Minmax																		
*m*TSP51	3	203	161	**160**	203	**160**	**160**	**160**	**160**	**160**	**160**	162	3	**160**	NA			
	5	164	119	**118**	154	**118**	**118**	**118**	**118**	**118**	**118**	120	1	**118**	NA			
	10	123	112	**108**	113	112	112	112	**112**	112	**112**	112	0	**112**	NA			
*m*TSP100	3	13,556	8542	8817	12,726	8636	8577	8569	**8509**	**8550**	**8509**	8571	28	**8509**	NA			
	5	10,589	6852	6964	10,086	6809	6785	6788	6768	**6769**	**6767**	6835	37	**6767**	NA			
	10	9463	6370	6363	7064	**6358**	**6358**	**6358**	**6358**	**6358**	**6358**	**6358**	0	**6358**	NA			
	20	8388	6359	6356	6402	**6358**	**6358**	**6358**	**6358**	**6358**	**6358**	**6358**	0	**6358**	NA			
*m*TSP150	3	19,687	13,268	13,885	18,019	14,074	13,896	13,697	13,461	**13,313**	**13,168**	13,628	189	13,376	NA			
	5	14,748	8660	9270	12,619	8977	8889	8781	8778	**8567**	8479	8601	62	**8467**	NA			
	10	11,158	5875	6132	8054	5885	5803	5815	5728	**5651**	**5594**	5736	39	5674	NA			
	20	10,044	5252	5250	5673	5270	5246	5248	**5246**	**5246**	**5246**	**5246**	0	**5246**	NA			
	30	8775	5247	**5246**	5270	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**	0	**5246**	NA			
Minsum																		
*m*TSP51	3	543	449	448	492	**446**	**446**	448	**446**	448	**446**	449	2	**446**	449	2	**446**	0.00
	5	586	479	478	519	475	475	475	**472**	478	**472**	475	0	474	**474**	1	**472**	0.21
	10	723	584	584	670	583	580	581	**580**	583	581	581	2	**580**	**580**	1	**580**	0.17
*m*TSP100	3	26,653	22,051	22,619	26,130	21,865	21,798	21,814	**21,798**	21,941	**21,798**	22,249	349	21,879	22,068	204	21,879	0.81
	5	30,408	23,678	24,166	28,612	23,352	23,238	23,221	23,182	**23,319**	23,294	24,706	694	23,294	23,383	206	**23,175**	5.35
	10	31,227	28,488	27,890	30,988	27,356	27,023	**27,004**	**26,961**	27,072	**26,961**	28,896	434	28,380	27,368	229	27,008	5.29
	20	54,700	40,892	39,949	44,686	40,484	39,509	38,397	38,333	**38,357**	**38,245**	40,327	234	39,764	38,867	447	38,326	3.62
*m*TSP150	3	47,418	38,434	39,247	44,674	38,385	38,276	38,262	38,066	**38,055**	**37,957**	39,053	395	38,543	38,827	237	38,430	0.58
	5	49,947	39,962	40,647	47,811	39,447	39,309	39,202	38,979	**38,881**	**38,714**	39,953	289	39,644	39,566	207	39,171	0.97
	10	54,958	44,274	44,436	51,326	43,323	43,038	42,712	42,441	**42,462**	**42,234**	43,816	298	43,264	42,922	198	42,703	2.04
	20	73,934	56,412	55,980	62,400	54,689	54,279	53,877	53,603	**53,612**	**53,475**	55,779	262	55,513	53,854	167	53,576	3.45
	30	99,547	72,783	71,266	78,023	69,480	69,048	69,045	68,865	**68,751**	**68,541**	71,073	640	70,421	68,804	151	68,558	3.19

NA: not applicable.

Fig. 3 presents the convergence (to the best known solution in the literature) results on the *m*TSP150 and $n = 128$ problems for both objective functions. The horizontal axis shows the running time allowed while the vertical axis shows the percent deviation from the best known obj. value. According to this figure the algorithm converges early and small improvements are observed after that time. Especially the *minsum* problems converge earlier than the corresponding *minmax* problems. It is very clearly observed from Fig. 3(b) that as the number of salesmen increases the convergence occurs earlier. We also observe that the *m*TSP150 problems converge before 200 s. while the $n = 128$ problems converge a little after 200 s. Remember that in all experiments the running time of the algorithm is set as *n* seconds, which is close to the convergence point according to these results.

According to the above settings, we run the GVNS algorithm and obtain the resulting tours given in Fig. 4 for the 5TSP100 problem. According to this figure, while the tour lengths are balanced in the *minmax* solution, one very long tour dominates the others in the *minsum* solution. This case is also observed by Bertazzi et al. (2015) in the worst case analysis of the *m*TSP where the *minsum* solution would be very poor if it is used for the solution of the *minmax* problem.

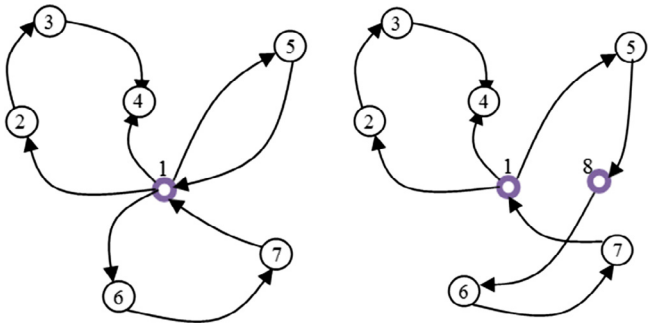
The performance of the initial heuristic is given in Figs. 5 and 6 for *minmax* *m*TSP150 and *minsum* *m*TSP 150 instances, respectively. When compared with the final objective functions, the performance of the initial heuristic is better in the *minmax* problem, and as the number of salesmen increases it closes to GVNS results indicating the success of the initial heuristic. However, the GVNS Seq-VND algorithm much improves the resulting solution of the initial heuristic in *minsum* problems.

As stated in the literature review, the nature inspired algorithms have been applied to the *m*TSP. In this section, we performed detailed tests on the proposed algorithm and compared its performance with algorithms from the literature. The results are presented in Tables 4 and 5 for first and second group of problems, respectively. Accordingly, ABC (FC), ABC (VC) and IWO represent the artificial bee colony algorithms and the invasive weed optimization algorithm (all with local search strategy) proposed by Venkatesh and Singh (2015), respectively. TCX represents the genetic algorithm with a two-part crossover operator proposed by Yuan et al. (2013). GGA-SS represents the grouping genetic algorithm by Singh and Baghel (2009). GA2PC represents the two-part chromosome representation based genetic algorithm proposed by Carter and Ragsdale (2006). ACO represents the ant colony algorithm developed by Liu et al. (2009). The best result of all algorithms in terms of *Avg.* and *best* measures are marked as bold. According to Table 4, the optimal results are also found by the GVNS Seq-VND algorithm for small size problems. In the instances of 128 nodes, the GVNS Seq-VND algorithm presents the best performance for *minmax* objective and considerably improves the best known results by the IWO algorithm. (The detailed tours of the best solutions by the GVNS for 128 nodes are provided in Supplementary material.) However, in the *minsum* objective its performance decreases. According to Table 5, the GVNS Seq-VND algorithm performs very close to the IWO algorithm for the *minmax* objective. However, the performance of the GVNS Seq-VND algorithm deteriorates especially in the large size instances of *minsum* problems. Next, we present a transformation scheme for the *minsum* objective, which improves the performance of the GVNS Seq-VND algorithm.

4.1. Transformation of minsum mTSP to minsum 2TSP problem

The studies have been conducted on transforming the *minsum* *m*TSP problem into an equivalent TSP (Svestka & Huckfeldt, 1973; Bellmore & Hong, 1974). One of the first transformations was by Svestka and Huckfeldt (1973), who suggested a transformation where the original distance matrix is augmented with $m - 1$ new rows and columns such that each new row and column is a duplicate of the first row and column of the original distance matrix. In this study, a similar transformation is performed in order to test the performance of the proposed algorithm. However, our algorithm is designed for $m > 1$ problems; any *m*TSP is transformed to an equivalent 2TSP by augmenting the distance matrix with $m - 2$ new rows and columns as stated above. This means that we are actually adding $m - 2$ dummy home depots to the problem. For instance, if the problem is 3TSP, then one dummy depot will be added into the problem and this dummy depot will exist in one of two tours in the transformed problem (see Fig. 7). Travel among depots is prohibited by penalizing the arcs between them. This type of transformation can also be applied to any heuristic approach and is expected to improve its performance. In our case, the GVNS algorithm solving this transformed problem is called the GVNS Seq-VND2.

The results of the GVNS Seq-VND2 algorithm are presented in the last columns of Tables 4 and 5. The %imp $\left(\frac{GVNS\ SeqVND.avg - GVNS\ SeqVND2.avg}{GVNS\ SeqVND.avg}\right)$ column gives the average deviation of the GVNS Seq-VND2 algorithm from the GVNS Seq-VND algorithm. Accordingly, the optimal results are also found by the GVNS Seq-VND2 algorithm and we observe an average 10% improvement in the $n = 128$ problem. The proposed GVNS algorithms present the best performance in Table 4. According to results given in Table 5, small improvements are observed with the transformed problem. Although the best performance is given by the GVNS Seq-VND2 algorithm in the *m*TSP51 data set, its performance is not best in the *m*TSP100 and *m*TSP150 data sets. We also compare both versions on asymmetrical TSP data sets (FTV35 (Reinelt, 1991) and TrN170 (Bayram, Peker, Katip, Akyurt, & Soylu, 2014)) with *minsum* objective. The results given in Table 6 reveal that the transformation does not affect the performance much in asymmetrical data sets.



Tours when $m = 3$ and $n = 7$ (node 1 is home depot)

Equivalent tours when $m = 2$ and $n = 8$ (node 8 is dummy home depot)

Fig. 7. Example 3TSP7 tours and equivalent 2TSP8 tours.

Table 6
Comparison of transformation results on asymmetric TSP instances.

Data set	*m*	GVNS Seq-VND			GVNS Seq-VND2			
		Avg.	SD	Best	Avg.	SD	Best	% Imp.
TrN 170	6	201.4	1	199.8	200.1	1	199.3	0.5
	20	239.4	1	238.4	236.4	0	235.8	1.3
FTV35	3	1525.3	15	1512.0	1527.0	27	1512.0	−0.1
	6	1652.4	22	1643.0	1648.2	12	1643.0	0.2

Table 7
Application results.

Tour no.	Current		GVNS Seq-VND (*minmax*)							GVNS Seq-VND (*minsum*)		GVNS Seq-VND2 (*minsum*)	
			Solution 1		*Solution* 2		*Solution* 3						
	Length (km)	#Of junctions	Length (km)	#Of junctions	Length (km)	#Of junctions	Duration (min)	Length (km)	#Of junctions	Length (km)	#Of junctions	Length (km)	#Of junctions
1	54.5	30	70.8	41	66.2	46	226.9	38.5	30	71.0	70	130.9	96
2	45.7	29	70.8	3	71.0	6	226.8	45.9	27	9.3	12	1.32	1
3	34.1	30	69.3	54	68.6	41	226.6	35.8	31	0.6	1	0.6	1
4	39.0	27	69.6	40	64.9	32	226.4	33.2	32	31.7	32	1.4	1
5	34.1	26	66.0	28	62.7	21	227.3	31.1	33	1.4	1	5.7	10
6	104.2	27	70.8	3	48.3	23	226.9	73.4	16	85.9	53	59.3	60
Max. tour length	104.2		70.8		71.0			73.4		85.9		130.9	
Total length	311.6		417.2		335.3			258		199.8		199.3	
Range	70.1		4.8		22.7			42.3		85.3		130.3	

4.2. Application

Finally, we applied the proposed algorithms to the traffic signalization network of Kayseri province in Turkey. In this network, there are 170 junctions, which include two to four traffic lamps to control the traffic at each junction, and a signalization team routinely checks them six days a week. During these checks the timing of light durations, clearness of traffic lights, correct working of basic equipments, availability of structural equipments, etc. are controlled and fixed if needed. Currently, these 170 junctions are divided into six tours and each junction is visited once a week. The tours start from the same depot. However, since the tours are not balanced, the team might have to work hard on some days and get behind of the daily schedule. As new junctions are added into the system, the problem becomes more complex and new junctions might be added into unrelated routes. Herein our aim is to minimize the maximum tour length. This problem is typically a *minmax* *m*TSP. The distance matrix is constructed through the GIS system and it is asymmetric. It is assumed that the travel speed is fixed and the checking duration at each junction is negligible. Bayram et al. (2014) collected the required data from the system in a vehicle routing study where they developed a four-stage heuristic. Unlike this study, they also considered uncertain travel times and checking durations. They evaluated different scenarios and tried to obtain a robust solution.

Table 7 presents the tour lengths and number of junctions available in each tour in the current situation and also presents the best *minmax* and *minsum* solutions found by the proposed algorithms. We observe that the current junction assignment strategy is to balance the number of nodes in each tour rather than tour lengths. However, this strategy results in a very long tour with a length of 104.2 km. Since the signalization team conducts a single tour every day, they also complain of work being over schedule when tour 6 is conducted. When we analyze the *minsum* results of the proposed algorithms, it is clear that these solutions are impractical since they are very unbalanced. According to ten replications of the *minmax* results, two solutions are highlighted with a very close maximum tour length. Although the tours in *solution* 1 are more balanced, the total tour length is 82 km. shorter in the second solution. Moreover, this gain is obtained just by sacrificing 0.2 km from the maximum tour length. In *solution* 2, one of the tours is shorter than the others. When we consider a light weekend working schedule, this solution brings another advantage. Accordingly, the maximum tour length improves 32% and the range improves 68% when compared to the current case. In *solution* 3, we assume that the travel speed is 30 km/h and the expected checking duration of each junction is 5 min. In this case, our aim is to minimize the maximum route duration. According to results, route durations are balanced but the route lengths deviate much. As can be seen, delays at nodes affect the solution in *minmax* problems. More realistic estimations of travel times and checking durations would bring the problem closer to the real life. As new junctions are added into the network, the inputs and the solution should be updated. A visual interface (under construction now) would then help to the team.

5. Conclusion

In this study, we consider the multiple traveling salesman problem with *minmax* and *minsum* objectives. We observed that only very small instances of the *minmax* problem can be optimally solved within a reasonable time limit although medium size instances of the *minsum* problem can be solved. A GVNS algorithm using various local search operators is proposed for the *m*TSP. Although it was designed for the *minmax* *m*TSP problems because of the motivation in our application, it was also adapted to the *minsum* *m*TSP. We have also presented the second version of the GVNS algorithm which can be used to solve the *minsum* *m*TSP, $m > 2$, where the problem is transformed into the *minsum* 2TSP. The performance of the proposed algorithms was evaluated and compared with that of other algorithms from the literature on test problems from the literature. The proposed algorithm performs better than other algorithms in *minmax* *m*TSP instances. In *minsum* *m*TSP instances, it performs close to the best known results. Our computational results also show that the transformation improves the performance, especially in symmetrical data sets.

We also applied the proposed GVNS algorithm for the traffic signalization network in the Kayseri province of Turkey where the tour lengths are currently very unbalanced. We obtained a significant improvement in terms of maximum tour length and range.

Another further research direction would be to take into account uncertainty in the *m*TSP. Using interval data may help to obtain robust solutions. Additionally, the tradeoff between *minmax* and *minsum*objective functions leads us to reevaluate the problem in the multi-objective context.

Acknowledgement

We would like to thank Prof. A.E. Carter, Prof. C.T. Ragsdale and Dr. M.S. Yuan for providing us with data sets and useful information. We also would like to thank the Kayseri Metropolitan Municipality for their help.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.cie.2015.10.010.

References

Bayram, R., Peker, H. H., Katip, H., Akyurt, F., & Soylu, B. (2014). *The vehicle routing study for traffic signalization teams and the Kayseri application*. Project report of 2209A program of The Scientific and Technological Council of Turkey (TÜBİTAK).

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega, 34*(3), 209–219.

Bellmore, M., & Hong, S. (1974). Transformation of multisalesman problem to the standard traveling salesman problem. *Journal of the ACM (JACM), 21*(3), 500–504.

Bertazzi, L., Golden, B., & Wang, X. (2015). Min–max vs. min–sum vehicle routing: A worst-case analysis. *European Journal of Operational Research, 240*(2), 372–381.

Brown, E. C., Ragsdale, C. T., & Carter, A. E. (2007). A grouping genetic algorithm for the multiple traveling salesperson problem. *International Journal of Information Technology & Decision Making, 6*(02), 333–347.

Carter, A. E., & Ragsdale, C. T. (2006). A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European Journal of Operational Research, 175*(1), 246–257.

Cordeau, J. F., Laporte, G., Savelsbergh, M. W., & Vigo, D. (2006). Vehicle routing. *Transportation, Handbooks in Operations Research and Management Science, 14*, 367–428.

Cplex, 2010. IBM ILOG Cplex 12.1 optimizer user's manual. <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer>.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research, 6*(6), 791–812.

França, P. M., Gendreau, M., Laporte, G., & Müller, F. M. (1995). The *m*-traveling salesman problem with minmax objective. *Transportation Science, 29*(3), 267–275.

Frederickson, G. N., Hecht, M. S., & Kim, C. E. (1978). Approximation algorithms for some routing problems. *SIAM Journal on Computing, 7*, 178–193.

Gavish, B. (1976). A note on "The formulation of the *m*-salesman traveling salesman problem". *Management Science, 22*(6), 704–705.

Gavish, B., & Srikanth, K. (1986). An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Research, 34*(5), 698–717.

Ghafurian, S., & Javadian, N. (2011). An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems. *Applied Soft Computing, 11*(1), 1256–1262.

Golden, B. L., Laporte, G., & Taillard, É. D. (1997). An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. *Computers & Operations Research, 24*(5), 445–452.

Groër, C., Golden, B., & Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation, 2*(2), 79–101.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449–467.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2008). Variable neighbourhood search: Methods and applications. *4OR, 6*(4), 319–360.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research, 175*(1), 367–407.

Hsu, C. Y., Tsai, M. H., & Chen, W. M. (1991). A study of feature-mapped approach to the multiple travelling salesmen problem. In *IEEE international symposium on circuits and systems, 1991* (pp. 1589–1592). IEEE.

Imran, A., Salhi, S., & Wassan, N. A. (2009). A variable neighborhood-based heuristic for the heterogeneous fleet vehicle routing problem. *European Journal of Operational Research, 197*(2), 509–518.

Kara, I., & Bektas, T. (2006). Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research, 174*(3), 1449–1458.

Kindervater, G. A., & Savelsbergh, M. W. (1997). Vehicle routing: Handling edge exchanges. *Local Search in Combinatorial Optimization*, 337–360.

Király, A., & Abonyi, J. (2011). Optimization of multiple traveling salesmen problem by a novel representation based genetic algorithm. In *Intelligent computational optimization in engineering* (pp. 241–269). Berlin Heidelberg: Springer.

Laporte, G., & Nobert, Y. (1980). A cutting planes algorithm for the *m*-salesmen problem. *Journal of the Operational Research Society*, 1017–1023.

Larki, H., & Yousefikhoshbakht, M. (2014). Solving the multiple traveling salesman problem by a novel meta-heuristic algorithm. *Journal of Optimization in Industrial Engineering, 16*, 55–63.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), 498–516.

Liu, W., Li, S., Zhao, F., & Zheng, A. (2009). An ant colony optimization algorithm for the multiple traveling salesmen problem. In *4th IEEE conference on industrial electronics and applications, 2009. ICIEA 2009* (pp. 1533–1537). IEEE.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM), 7*(4), 326–329.

Mladenović, N., Urošević, D., & Hanafi, S. (2013). Variable neighborhood search for the travelling deliveryman problem. *4OR, 11*(1), 57–73.

Mladenović, N., Urošević, D., & Ilić, A. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research, 220*(1), 270–285.

Modares, A., Somhom, S., & Enkawa, T. (1999). A self organizing neural network approach for multiple traveling salesman and vehicle routing problems. *International Transactions in Operational Research, 6*(6), 591–606.

Na, B. (2006). *Heuristic approaches for the no-depot k-traveling salesmen problem with minmax objective*. Master's thesis, Texas, USA: Texas A&M University; May 2006.

Polat, O., Kalayci, C. B., Kulak, O., & Günther, H. O. (2015). A perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and delivery with time limit. *European Journal of Operational Research, 242*(2), 369–382.

Potvin, J. Y., Lapalme, G., & Rousseau, J. M. (1989). A generalized *k*-opt exchange procedure for the MTSP. *INFOR, 27*(4), 474–481.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing, 3*(4), 376–384.

Russell, R. A. (1977). Technical note—An effective heuristic for the *m*-tour traveling salesman problem with some side conditions. *Operations Research, 25*(3), 517–524.

Salhi, S., & Rand, G. K. (1987). Improvements to vehicle routeing heuristics. *Journal of the Operational Research Society*, 293–295.

Sarin, S. C., Sherali, H. D., Judd, J. D., & Tsai, P. F. J. (2014). Multiple asymmetric traveling salesmen problem with and without precedence constraints: Performance comparison of alternative formulations. *Computers & Operations Research, 51*, 64–89.

Sedighpour, M., Yousefikhoshbakht, M., & Mahmoodi Darani, N. (2012). An effective genetic algorithm for solving the multiple traveling salesman problem. *Journal of Optimization in Industrial Engineering* (8), 73–79.

Singh, A., & Baghel, A. S. (2009). A new grouping genetic algorithm approach to the multiple traveling salesperson problem. *Soft Computing, 13*(1), 95–101.

Somhom, S., Modares, A., & Enkawa, T. (1999). Competition-based neural network for the multiple travelling salesmen problem with minmax objective. *Computers & Operations Research, 26*(4), 395–407.

Song, C. H., Lee, K., & Lee, W. D. (2003). Extended simulated annealing for augmented TSP and multi-salesmen TSP. *Proceedings of the international joint conference on neural networks, 2003* (Vol. 3, pp. 2340–2343). IEEE.

Svestka, J. A., & Huckfeldt, V. E. (1973). Computational experience with an *m*-salesman traveling salesman algorithm. *Management Science, 19*(7), 790–799.

Tang, L., Liu, J., Rong, A., & Yang, Z. (2000). A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel complex. *European Journal of Operational Research, 124*(2), 267–282.

Venkatesh, P., & Singh, A. (2015). Two metaheuristic approaches for the multiple traveling salesperson problem. *Applied Soft Computing, 26*, 74–89.

Wacholder, E., Han, J., & Mann, R. C. (1989). A neural network algorithm for the multiple traveling salesmen problem. *Biological Cybernetics, 61*(1), 11–19.

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, 16*(1), 65–80.

Yuan, S., Skinner, B., Huang, S., & Liu, D. (2013). A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. *European Journal of Operational Research, 228*(1), 72–82.

Yu, Q., Wang, D., Lin, D., Li, Y., & Wu, C. (2012). A novel two-level hybrid algorithm for multiple traveling salesman problems. In *Advances in swarm intelligence* (pp. 497–503). Berlin Heidelberg: Springer.

Zhao, W., Chen, H., & Li, H. (2012). A variable neighborhood search approach for multiple traveling salesman problem with deadlines. In *2012 9th IEEE international conference on networking, sensing and control (ICNSC)* (pp. 301–306). IEEE.