Journal of Project Management 6 (2021) 209–222

Contents lists available at GrowingScience

Journal of Project Management

homepage: www.GrowingScience.com/jpm

Solving open travelling salesman subset-tour problem through a hybrid genetic algorithm

Purusotham Singamsetty[a], Jayanth Kumar Thenepalle[b*] and Balakrishna Ururu[b]

[a]*Department of Mathematics, School of Advanced Sciences, VIT, Vellore-632014, Tamil Nadu, India*
[b]*Department of Science and Humanities, Sreenivasa Institute of Technology and Management Studies, Chittoor-517127, Andhra Pradesh, India*

C H R O N I C L E

Article history:
Received: November 30, 2020
Received in revised format: April 2, 2021
Accepted: May 2, 2021
Available online: May 5, 2021

Keywords:
Travelling salesman problem
Open travelling salesman subset-tour problem
Genetic algorithm
Complex mutation

A B S T R A C T

In open travelling salesman subset-tour problem (OTSSP), the salesman needs to traverse a set of k ($\leq n$) out of n cities and after visiting the last city, the salesman does not necessarily return to the central depot. The goal is to minimize the overall traversal distance of covering k cities. The OTSSP model comprises two types of problems such as subset selection and permutation of the cities. Firstly, the problem of selection takes place as the salesman's tours do not contain all the cities. On the other hand, the next problem is about to determine the optimal sequence of the cities from the selected subset of cities. To deal with this problem efficiently, a hybrid nearest neighbor technique based crossover-free Genetic algorithm (GA) with complex mutation strategies is proposed. To the best of the author's knowledge, this is the first hybrid GA for the OTSSP. As there are no existing studies on OTSSP yet, benchmark instances are not available for OTSSP. For computational experiments, a set of test instances is created by using TSPLIB. The extensive computational results show that the proposed algorithm is having great potential in achieving better results for the OTSSP. Our proposed GA being the first evolutionary-based algorithm that will help as the baseline for future research on OTSSP.

1. Introduction

The travelling salesman problem (TSP) is one of the broadly considered combinatorial optimization routing problems that was first coined by two mathematicians in the 1800s (Matai et al., 2010), and then formulated by Karl Menger (Maredia, 2010). In TSP, a set of cities and a salesman will be given. The salesman is intended to cover all the given cities and come back to the city where he started. The objective is to cover all the given cities and come back to the home city with overall least traversal distance. However, the basis of the TSP is aimed to determine a closed Hamilton path that means a path that covers every node/city in the graph exactly once and comes back to the starting point, the works on Open Travelling Salesman-Subset Tour Problem (OTSSP) is limited. Yet, in the practical transportation scenarios, the salesman need not necessarily cover all the given n cities, but only enough to cover k cities from the given n cities. The OTSSP can be described as follows: Given a set of n cities including a starting/home/depot city and a predetermined value k ($\leq n$), the OTSSP aims to find a subset of k cities having the depot city, where the salesman starts the tour, cover each city from this subset exactly once and not necessarily to come back to the depot city such that the overall traversal distance. In other words, the OTSSP consists of selecting a subset with exactly k out of n cities and sequencing them to minimize the overall traversal distance by the salesman. This model has wide practical utility in those situations like when there are inadequate resources to cover all the given cities and the organizations working together with outsourcing agencies. More often, this scenario can be seen in rural healthcare servicing and design of distribution networks. For instance, if the logistic distribution agency taking services from an outsourcing agency is not having sufficient resources to cover all the cities, it is significant to serve partial cities. Thus,

* Corresponding author.

E-mail address: jayanth.maths@gmail.com (J. Thenepalle)

doi: 10.5267/j.jpm.2021.5.002

the salesman starts from the depot city, covers only a limited number of cities instead of all the cities and need not return to the home city at the end with the least possible distance. Figures 1, 2 and 3 demonstrate the difference between classical travelling salesman problem (TSP), travelling salesman subset tour problem (TSSP) and open travelling salesman subset tour problem (OTSSP), respectively. Fig. 1 represents the classical TSP in which the salesman starts from a depot city, covers the rest of the given 7 cities exactly once and comes back to the home city with minimum overall traversal distance. Fig. 2 depicts an arbitrary solution of TSSP in which the salesman starts from the depot city, needs to cover only *k=6* out of 8 cities and returns to the depot city. Finally, Fig. 3 indicates an arbitrary solution of OTSSP, where the salesman starts from the depot city, needs to cover only *k=6* out of 8 cities and not necessarily to come back to the depot city.

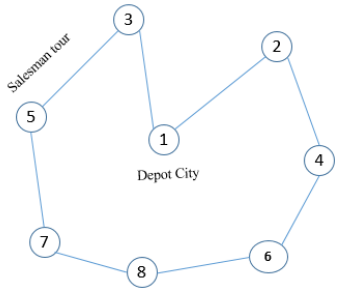


Fig. 1. An example solution of TSP with $n=8$ cities

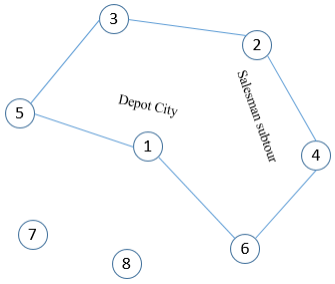


Fig. 2. An example solution of TSSP with $k=6$ cities

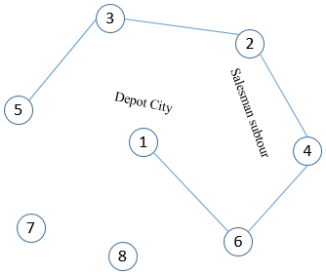


Fig. 3. An example solution of OTSSP with $k=6$ cities

The classical TSP can be seen as a special case of TSSP with *k=n* and can be seen as OTSSP in the presence of closed tour and *k=n*. On the other hand, TSSP in itself can be seen as an exceptional case of prize collecting travelling salesman problem (PCTSP) (Balas, 1989) in which each city is specified with a prize as well as a penalty. When the salesman visits any city, he will accumulate the prize corresponding to the visited city and if he does not visit any city, then a penalty will be incurred related to the unvisited city. The objective of the PCTSP is to minimize the overall traversal distance covered by the salesman and the net penalties acquired while accumulating a specified minimum total of the prize. When penalties for every city is assumed to be zero, the PCTSP becomes the quota travelling salesman problem (QTSP). When we assume the prize assigned to each city is 1 and the specified minimum total of the prize is *k*, then the model QTSP can turn into TSSP (Ausiello et al., 2018). Since the classical TSP is NP-hard, it is obvious that the variants namely TSSP and OTSSP are also NP-hard. To best of author's knowledge, Chieng & Wahid, (2014) first addressed and developed a genetic algorithm to solve OTSSP model. We address the same problem OTSSP and develop a hybrid nearest neighbour based genetic algorithm. Reviewing the literature, the work on OTSSP is still very limited. The TSSP is a special case of OTSSP, where the salesman need not necessarily return to the starting city. Concerning the literature, the TSSP is also referred to as *k*-TSP (Venkatesh et al., 2018). To review the earlier works, Saksena and Kumar (1966), Ibaraki (1973) and Laporte et al. (1984) studied TSSP with an additional constraint that the salesman has to visit a subset of *k(≤n)* specified nodes. A variety of solution techniques including heuristics and exact algorithms for TSSP and its variants can be found in the literature. Gensch (1978) modelled TSSP as an industrial scheduling problem with an additional time constraint. In this variant, the salesman needs to cover only a subset of the cities from a given set of cities with minimum distance without violating the time limit. To solve this problem, the Lagrangian relaxation-based branch and bound algorithm have been developed that deals problems of a practical size. Mittenthal and Noon (1992) proposed an insertion/ deletion based heuristic algorithm to solve TSSP with an additional constraint efficiently and its effectiveness is shown through experimental results. Verweij and Aardal (2003) discussed the merchant sub tour problem whose objective is to find the closed tour that maximizes the profit over a vertex and edge-weighted complete graph and developed a linear programming approach for its solution. Westerlund et al. (2006) addressed TSSP that finds a path from a depot on the node and edge-weighted undirected graph with prizes and Knapsack limitations on the node weights. A heuristic decomposition scheme-based column generation algorithm was developed to solve this problem.

Giardini and Kalmár-Nagy (2011) studied sub tour problem and it has been applied to multiagent planning scenarios. To tackle this model, a hybrid Genetic algorithm (GA) combined with a local search algorithm has been developed and has demonstrated the algorithm's efficiency through experimental results. Stetsyuk (2016) has given the statement of *k*-node shortest cycle in a complete weighted graph and formulated as a Boolean linear programming problem. This study has experimentally shown that the problem of determining the *k*-node shortest cycle is more challenging than the problem of identifying the shortest Hamiltonian cycle. This is due to the solution consisting in finding the optimal subset of nodes corresponding to the shortest Hamiltonian cycle. Venkatesh et al. (2018) considered *k*-TSP and proposed a first simple and efficient metaheuristic algorithm called general variable neighborhood search algorithm (GVNS) to solve TSSP/ *k*-TSP. This technique combines two neighborhood strategies such as exchange and swap processes, which effectively accomplish both the features such as subset selection and permutation of the cities of the *k*-TSP. More recently, Venkatesh et al. (2020) has developed two multi-start heuristic algorithms, which combines a GVNS approach and hyper-heuristic for solving *k*-TSP. From the literature, of all the metaheuristic approaches, Genetic algorithm (GA) has proven to have good capability in dealing combinatorial optimization problems (Bahaabadi et al., 2012). The former-cited works motivate to address the practical variant of TSP called

OTSSP. Solving the OTSSP includes two features, namely subset selection and permutation. Here, subset selection is itself a problem of picking a subset that contains *k* out of *n* cities with depot city as well and permutation means the problem of determining the best permutation of the *k* cities from the selected subset. Solution methods for OTSSP have to address both of these features in a suitable way to tackle a wide range of test instances effectively. If the approach to deal with the subset selection feature is not project appropriately, it will not produce the best solutions for OTSSP and it does not matter how best the approach is designed for permutation feature. Similarly, an approach with a good scheme to tackle permutation features but a weak scheme for subset selection feature, will not yield the best solutions either. Hence, for any solution approach to solve OTSSP, relative importance should be given to these two aspects to getting the best solutions. By understanding its importance, we have developed a hybrid nearest neighbor technique based Genetic algorithm with complex mutation operators for OTSSP. The main contribution of this study is addressing a practical variant namely OTSSP and development of a hybrid Genetic algorithm (GA) that effectively solves OTSSP. Generally, the initial population is produced randomly. Since the OTSSP solutions do not include all the cities, GA through randomly generated population may result in solutions far away from the optimal or near-optimal solutions. Hence, an efficient systematic procedure is essentially required to generate the initial population. However, in this study, we propose an efficient nearest neighborhood algorithm that effectively generates relatively good initial population. Further, to make the GA converge toward optimal or near-optimal solutions, complex mutation operators that include a slide, swap, reverse swap, and other combinations being considered. To the best of author's knowledge, the proposed GA is the first evolutionary hybrid algorithm for OTSSP.

The rest of the paper is organized as follows: The following section will provide the definition and formulation of the OTSSP. Section 3 will give an outline of the GA and its operators. Section 4 demonstrates the computational results. Finally, the conclusion and scope of future work are described in Section 5.

2. Problem statement and mathematical formulation

Let $G=G(V,E)$ be the complete, undirected, edge-weighted graph, where the node-set $V=\{1,2,...,n\}$ denotes a set of n cities including one central depot /home city/starting city and the edge set $E=\{(i,j)/i,j\in V;\ i\neq j\}$ be the set of n^2-n edges. Note that the terms node and city are synonymously used hereafter. Each node is specified with a position (x_i,y_i) in the Cartesian coordinate system. Each edge (i,j) is associated with a distance/cost $d_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$; $(d_{ij}=d_{ji};d_{ii}=\infty,d_{ij}>0)$, which is the Euclidean distance between the cities i and j. Let the salesman be positioned at the starting city/ central depot. The salesman need to cover a subset $S(|S|=k,where\ \ S\subseteq V)$ of $k(\leq n)$ cities starts from the home city, takes a route by covering the rest of the $k-1$ cities of S exactly once and do not necessarily return to the home city. The possible sub tours induced on covering these k cities will be $\binom{n}{k}\times(k-1)!$. The OTSSP aims to determine an optimal open path that covers k out of n cities, such that the overall traversal distance is minimized. Note that, a path of length k is said to be a feasible solution and an infeasible solution, otherwise. Here, the binary variable $x_{ij}\in\{0,1\}$, such that $x_{ij}=1$ if the salesman visits j^{th} city from i^{th} city, and $x_{ij}=0$, otherwise. Here, another binary variable $y_j\in\{0,1\}$ is introduced, such that $y_j=1$, if the j^{th} city is included in the subset and $y_j=0$, otherwise. Note that the starting city is assumed as city 1 in this study. The mathematical model for OTSSP is as follows:

$$\min\ Z=\sum_{i=1}^{n}\sum_{j=1}^{n}d_{ij}\,x_{ij} \tag{1}$$

subject to:

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

$$\sum_{i=2}^{n}x_{i1}=0 \tag{3}$$

$$\sum_{i=1}^{n}x_{ij}\leq 1;\ \forall\ j\in V\ \&\ i\neq j \tag{4}$$

$$\sum_{j=1}^{n} x_{ij} \leq 1;\ \forall\ i \in V\ \&\ i \neq j \tag{5}$$

$$x_{1p_1} + x_{p_1p_2} + x_{p_2p_3} + \ldots + x_{p_{i-1}p_i} = k-1;\ p_1, p_2 \ldots p_i \in V / \{1\} \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} - \sum_{p=2}^{n} x_{jp} \leq 1;\ \forall\ j \in V \tag{7}$$

$$\sum_{i=1}^{n} y_i = k \tag{8}$$

$$x_{ij}, y_i \in \{0,1\};\ \forall (i,j) \in E;\ i \in V \tag{9}$$

The objective function (1) represents the minimization of the overall traversal distance on touring *k* cities by the salesman. Constraint set (2-3) indicates that the salesman starts from the home city and not necessarily returns to the home city. Constraint set (4-5) ensures that the salesman can visit a city and departs from that city at most once. A Hamiltonian path with length *k-1* involves precisely *k-1* edges and *k* cities, thus the Constraint (6) is enforced to guarantee that there are *k-1* edges. However, this constraint does not guarantee the construction of the feasible path with those *k-1* edges (For instance, if $n = 9$ and *k=4*, one can select the 3-edges as (1,3), (3,2), (6,5), which cannot build a feasible path of length 3. Furthermore, the cities involved in the subset $S = \{1, 3, 2, 5, 6\}$ violate the desired cardinality i.e. $|S| = k \neq 4$. The trip must be a continuous path that starts at the home city and should cover exactly 4 cities including the home city. The degree of each city must be two (except the last city) in the path (one in degree and one out-degree). To maintain this, Constraint (7) has been introduced. Hence, it preserves the degree constraint for each city, except the last city in the path. Constraint (8) represents a feasible tour that covers exactly *k* cities. Finally, in Constraint (9), x_{ij} and y_j represents the decision variables that take binary values.

3. Genetic algorithm

In this section, first, the basic Genetic algorithm (GA) is described, and then the proposed algorithm is discussed in detail. The GA is one of the extensively used metaheuristic algorithms in evolutionary computation for solving combinatorial optimization problems (Goldenberg, 1989). This algorithm was first coined by Holland in 1975, which is an adaptive searching technique based on the survival of the fittest strategy. In its nature, the GA starts with a set of initial solutions/individuals called the initial population, also referred to as chromosomes, in which all the genetic data is stored. Each number within the chromosome is considered as a gene. Further, a fitness value is determined to evaluate the performance of a chromosome. Each time, two chromosomes, called parent chromosomes are selected from the population randomly, which is proportionate to their fitness value. Then, the two chromosomes perform crossover to produce two new chromosomes for the subsequent generation. These new chromosomes will swap old ones if they have superior fitness values. Then, a mutation operation is applied to the newly generated chromosomes to preserve the diversity of the population. Repeat selection, crossover, and mutation processes to generate more chromosomes that are new until the newly generated population size equals the old one. The iteration then starts with the new population. Since better chromosomes will always have a higher chance of being selected for crossover and the new chromosomes generated to transmit the characteristics of their parent chromosomes. The search process continues for many generations until stopping criteria are met. Thus, the entire process is called classical GA. However, there are certain studies in which GA without crossover has been developed. For instance, Liu & Kroll, (2016) studied multi-robot task allocation problem and a crossover-free GA with complex mutation operators (slide, inversion, swap, insertion, and other combinations) has been presented and showed that the crossover-free GA finds better results than that of the classical GA. To solve OTSSP via GA effectively, the key elements such as chromosome representation, population initialization, fitness evaluation, selection, crossover, mutation operators and GA parameters are required. Different GA strategies have distinct encoding, crossover and mutation operators, which results in divergence of the search process. Thus, it is inevitable to redesign the above operations to confirm that the optimal/suboptimal solution is indeed achieved. To solve OTSSP effectively, a crossover-free GA with complex mutation operators (swap, slide, reverse swap) is developed. The key elements involved in the proposed GA for solving OTSSP are described in the following subsections.

3.1. Encoding

The practice of genetic encoding is significant for producing feasible chromosomes. The strategies for encoding chromosomes vary from problem to problem and consists of a certain extent of art. For the travelling salesman problem (TSP), the solution is often indicated as a chromosome of length with the cities involved in the problem. Reviewing the literature, path representation is widely used in solving TSP and its variants (Hussain et al., 2017). In path representation, each chromosome is expressed by an arrangement of *n* distinct integers. To represent OTSSP solution, the present study utilizes a modified path representation in which a chromosome consists of *k-1* genes alone (since OTSSP solution involves only k cities). A chromosome can often be represented as $(g_1, g_2, g_3, \ldots g_{k-1})$, where $g_j \in V / \{\text{home city}\}, 1 \leq j \leq k-1$ and each g_i indicates a gene (city) in the chromosome. This kind of chromosome representation with length k is easy to implement and interpret as

OTSSP solution. For instance, if $n = 8$, $k = 6$, then an arbitrary OTSSP solution corresponding chromosome is $(7, 2, 4, 6, 3)$ and same is shown in Fig. 4. The resultant feasible path $1-7-2-4-6-3$ can be returned from the chromosome by affixing the home city at the beginning.

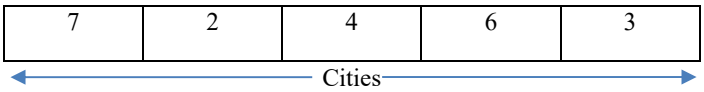


Fig. 4. An arbitrary OTSSP solution with 6 out of 8 cities

3.2. Initial population

A finite collection of feasible chromosomes is generally called the initial population and its generation plays a vital role in the GA. This pool consists of valid chromosomes, which are usually generated randomly. Since each of OTSSP solutions involves only *k* out of *n* cities, therefore identifying which *k* cities would result in the optimal or near-optimal solution is still a challenging task. Hence, a well-sophisticated technique is essentially needed to generate the initial population. In this study, the initial population is efficiently generated to assure better and faster convergence in producing the optimal or near-optimal results. The initial population for the proposed GA was created using the nearest neighbor algorithm. It is obvious that the best solution chromosome certainly includes minimum distance edges. With this fact, first, the elements of the distance matrix will be sorted ascending order along with their corresponding indices. By selecting the least distance corresponding first node and using the nearest neighbor heuristic, first, the chromosome is generated. In the same way, select the next least distance corresponding to the first node and apply the nearest neighbor heuristic, the second chromosome is produced. In such a way, for $n \times n$ symmetric matrices, a set of $(n^2-n)/2$ valid non-redundant chromosomes can be generated. Of which, only the desired number of best chromosomes can be chosen for further process. The pseudo-code of the nearest neighbor heuristic is presented in Algorithm 3.1. The chromosome demonstrated in Fig. 4, represents a single individual of the population called one of the solutions of the problem.

Algorithm 3.1. Nearest neighbor approach

begin Nearest_Neighbor
Initialization
A $n \times n$ distance matrix
Sort the elements of the distance matrix along with respective indices,
route= $\varnothing$
Generate initial population with nearest-neighbor heuristic
  while termination condition not met **do**
    find the least distance corresponding nodes
    select the first node as the current city from the two nodes
    start city $\longrightarrow$ current city
    route=route $\cup$ current city
    mark the current city as visited
    nearest (current city) $\longrightarrow$ next city
    If route length=desired length, **end**.
    else, nearest city $\longrightarrow$ current city
  end
Output salesman route with desired length
end Nearest_Neighbor

3.3. Fitness function

The fitness function helps to evaluate candidate chromosomes in the population. In our study, the fitness function is considered as the objective function given in Eq. (1). Therefore, the chromosome with smaller distance/cost will have a higher fitness value and thus have a greater genetic probability to be selected. For the OTSSP, the fitness value represents the overall distance of the salesman on covering exactly *k* cities including the home city.

3.4. Selection

The selection operator is another significant step in the GA, which helps to create a new population with higher fitness value from the current population. Its main intention is to carry the high-quality genes to the subsequent generation and to improve the efficiency of evaluation and convergence towards the optimal and near-optimal solution. The present study uses the

classical roulette wheel strategy to the selection operation. This operator selects a chromosome from its population in a statistical fashion depending on its fitness value to enter into a reproducing pool. Those chromosomes closer to the solution have a better chance of being selected.

3.5. Mutation operator

Mutation operation is performed next to the selection. It avoids the GA from being trapped in a local optimum and enhances the genetic variability of the population. This work utilizes the complex mutation operator, which comprises Swap, Reverse swap/Flip and Slide mechanisms. All these mutation operators are incorporated to get optimal solutions or near-optimal solutions in a limited time. With a mutation probability P_m, a parent chromosome is chosen. For a swapping operation, two different positions are chosen randomly from the parent chromosome; the genes of these two positions are interchanged. For a reverse swap operation, two different positions are chosen to describe segments, the genes between these positions are reversed. Similarly, for a slide operation, two distinct positions are selected (say, i^{th} and j^{th} positions). The new offspring can be produced by removing the gene in i^{th} position and copy the same in j^{th} position of the parent chromosome. Thus, genes between i^{th} and j^{th} positions will be decremented by one, i.e. the gene at $(i+1),(i+2)$ positions will be moved to i^{th} and $(i+1)^{th}$ positions, respectively and so on. Similarly, the gene at i^{th} position will be moved to j^{th} position and the gene associated at j^{th} position should be moved to $(j-1)^{th}$ position. Examples for Swap, Reverse swap/Flip and Slide operations are illustrated in Figs. 5-7, respectively.

3.6. GA parameters

In addition to the key elements of GA discussed earlier, setting appropriate values to the parameters namely, size of the population, mutation probability rate and termination criteria also plays a vital role in the algorithm's efficiency. These parameters are varied by the problem to be tackled. The population size indicates the number of chromosomes in any one generation and it is considered sufficient as large as 100 in this study. Although crossover operators are not considered in this study, with complex mutation strategies the diversity in the population can be achieved. Mutation probability rate (P_m) indicates how frequently the mutation operation is performed to the parts of the chromosome. It makes changes in the part of the chromosomes and thus maintains the diversity in the population. Generally, P_m lies between 0.001 and 0.1. In our study, it is considered as 0.01. Finally, the termination criterion of the GA is assumed to be a maximum number of generations. The process of the proposed GA is demonstrated in Fig. 8.

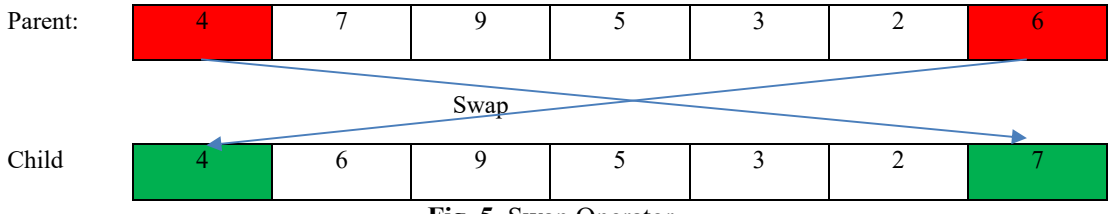


Fig. 5. Swap Operator

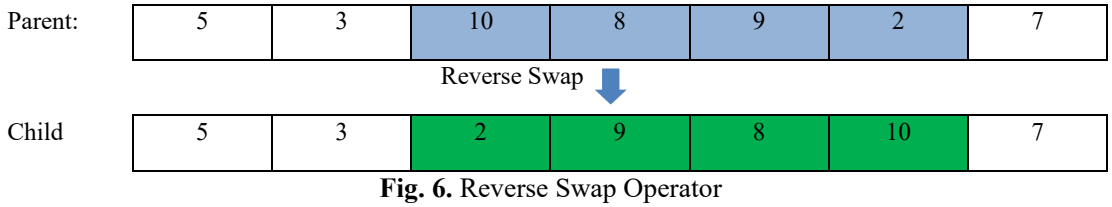


Fig. 6. Reverse Swap Operator

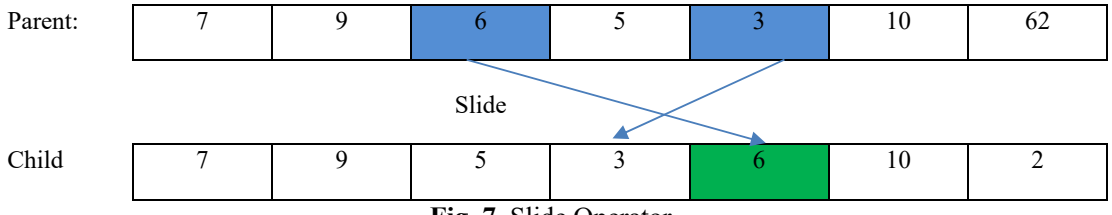


Fig. 7. Slide Operator

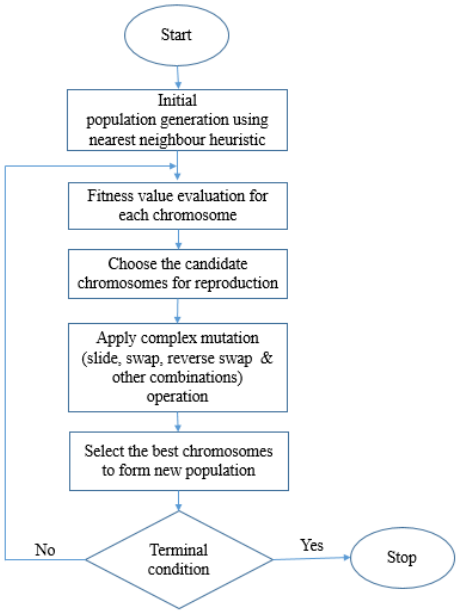


Fig. 8. Flowchart of the proposed GA

4. Computational results

Computational results are presented in this section. For all the experiments, the proposed GA uses roulette wheel selection, complex mutation (Swap, Reverse Swap and Slide) operators, to produce new offspring in every generation. The proposed GA is coded in MATLAB R2017a on a PC with Intel Core CPU i3 2.00 GHz and 4GB of Ram with Windows 10 Pro 64 bit like an Operating System. Since our nearest neighbor technique based genetic algorithm is the first solution method for OTSSP, no benchmark test instances are presented in the literature. Thus, the benchmark instances available in TSPLIB are utilized to create OTSSP instances. Overall, 50 instances have been considered from TSPLIB. These test instances were Euclidean, two-dimensional symmetric with distinct node scales, which are ranging from 14 to 318 cities. In all these instances, the first city is considered to be the home city. With each of these 50 instances, three distinct scenarios with each having a definite value for k (i.e. $k=\left\lfloor \frac{n}{4} \right\rfloor$, $k=\left\lfloor \frac{n}{2} \right\rfloor$, and $k=\left\lfloor \frac{3*n}{4} \right\rfloor$) are considered. This results in 150 test instances for the OTSSP. A comparative study is carried out on these instances. To do this, the proposed GA has been modified that fits for solving TSSP and tested all these 150 instances, best-found TSSP solutions are reported. However, the OTSSP is not the same as the TSP, typical test problems and optimal results of these may be useful to assess the proposed GA performance. To measure the performance of the proposed GA, each test instance is tested ten times independently and reports the best and worst solutions over ten independent runs. All the results were reported within the time limit of fewer than 10 minutes. Tables 1, 2 and 3 report the computational results of proposed GA executed on 150 test instances. In all these tables, the first column labelled *Instance* denotes the name of the test instance followed by the number of cities at the end. The second and third columns labelled n and k, respectively denote the size of the test instance (i.e. the number of cities involved in the test instance) and the number of cities required to be covered by the salesman (i.e. k out of n). The fourth and fifth columns respectively denote the best known TSSP solution presented in the literature and the best found TSSP solution through proposed GA. The sixth column labelled *Gap* denotes the gap/deviation between the best-known TSSP and best TSSP solution produced by proposed GA. It is evaluated by using the formula (10). Here the *Gap* takes both positive as well as negative values. Note that positive gap values specify that the best-found TSSP solution by proposed GA is better than the best-known TSSP solution and the value zero represent that both the solutions coincide. Similarly, negative values indicate that the best-found TSSP solution is worse than the best-known TSSP solution. Finally, columns seventh and eight labelled *Best* and *Worst* denotes the best and worst OTSSP solution produced by proposed GA, respectively.

$$Gap=\left|\frac{Best\ known\ TSSP\ solution-Best\ found\ TSSP\ solution\ by\ GA}{Best\ known\ TSSP\ Solution}\right| \tag{10}$$

Table 1

Computational results of proposed GA on instances with $k=\left\lfloor \frac{n}{4} \right\rfloor$

Instance	*n*	*k*	Best known TSSP solution	Best found TSSP solution	Gap	OTSSP solution by proposed GA	
						Best	Worst
a280	280	70	670	686	-0.0239	606	606
bayg29	29	7	332	332	0	246	246
bays29	29	7	400	400	0	282	282
berlin52	52	13	679	679	0	489	489
bier127	127	31	10619	10619	0	9840	10132
burma14	14	3	359	**280**	0.2201	151	151
ch130	130	32	1130	1130	0	1116	1119
ch150	150	37	1276	1294	-0.0141	1204	1204
d198	198	49	5027	**5002**	0.005	3269	3269
dantzig42	42	10	145	145	0	99	99
eil101	101	25	107	107	0	101	101
eil51	51	12	82	82	0	71	71
eil76	76	19	102	102	0	99	99
fri26	26	6	243	243	0	145	145
gil262	262	65	540	540	0	509	509
gr137	137	34	17399	17399	0	14784	14784
gr17	17	4	234	234	0	143	143
gr21	21	5	324	324	0	178	178
gr24	24	6	264	264	0	231	231
gr48	48	12	874	874	0	558	558
gr96	96	24	10460	**9543**	0.0877	7704	7704
gr202	202	50	8142	8142	0	6977	6977
gr229	229	57	18555	18555	0	18471	18471
hk48	48	12	2827	2827	0	2094	2094
kroA100	100	25	4970	5203	-0.0469	4369	4369
kroA150	150	37	5690	5690	0	5286	5286
kroA200	200	50	6202	6202	0	6138	6138
kroB100	100	25	4305	**4303**	0.0005	4014	4014
kroB150	150	37	5812	5812	0	5119	5119
kroB200	200	50	6368	**6100**	0.0421	5890	5890
kroC100	100	25	4964	4967	-0.0006	4293	4293
kroD100	100	25	4762	4762	0	3991	3991
kroE100	100	25	3905	3905	0	3663	3663
lin105	105	26	2606	2606	0	2108	2108
lin318	318	79	8901	8901	0	8705	8705
pr107	107	26	8443	8443	0	6981	6981
pr124	124	31	14640	**14325**	0.0215	9596	9596
pr136	136	34	21116	21116	0	20928	20928
pr144	144	36	14327	14327	0	10743	10743
pr152	152	38	23195	**20029**	0.1365	15403	15403
pr76	76	19	23450	23450	0	17728	17728
pr226	226	56	20033	20033	0	18594	18645
rat195	195	48	557	565	-0.0144	538	538
rat99	99	24	284	291	-0.0246	260	260
rd100	100	25	1438	1438	0	1261	1261
St70	70	17	120	120	0	110	114
Swiss42	42	10	192	**100**	0.4792	66	66
U159	159	39	8983	9085	-0.0114	8494	8494
Ulysses16	16	4	935	935	0	618	618
Ulysses22	22	5	747	747	0	447	447

Table 2

Computational results of proposed GA on instances with $k = \left\lfloor \frac{n}{2} \right\rfloor$

Instance	n	k	Best known TSSP solution	Best found TSSP solution	Gap	OTSSP solution by proposed GA	
						Best	Worst
at280	280	140	1314	1358	-0.0335	1234	1354
bayg29	29	14	626	626	0	581	581
bays29	29	14	733	733	0	672	672
berlin52	52	26	1874	1874	0	1766	1766
Bier127	127	63	26062	26107	-0.0017	24862	24862
burma14	14	7	1272	**1236**	0.0283	842	842
ch130	130	65	2408	2408	0	2492	2492
ch150	150	75	2761	2761	0	2772	2790
d198	198	99	7058	7086	-0.0040	5220	5261
dantzig42	42	21	260	260	0	227	227
eil101	101	50	227	227	0	249	249
eil51	51	25	175	175	0	179	181
eil76	76	38	216	219	-0.0139	217	219
fri26	26	13	414	414	0	308	339
gil262	262	131	1042	1042	0	1049	1049
gr137	137	68	29363	31784	-0.0825	29108	29108
gr17	17	8	517	517	0	359	368
gr21	21	10	918	918	0	683	683
gr24	24	12	504	504	0	396	396
gr48	48	24	1819	1819	0	1691	1691
gr96	96	48	20688	**19876**	0.0392	17634	17634
gr202	202	101	14181	14181	0	13996	14131
gr229	229	114	41005	41005	0	41661	41877
hk48	48	24	4701	4701	0	4238	4300
kroA100	100	50	9184	10050	-0.0943	9073	9098
kroA150	150	75	11625	11625	0	11412	11483
kroA200	200	100	12753	12753	0	13315	13315
kroB100	100	50	9096	9096	0	9071	9071
kroB150	150	75	11535	11535	0	11501	11501
kroB200	200	100	13080	13113	-0.0025	12787	12787
kroC100	100	50	9457	9457	0	9428	9498
kroD100	100	50	8719	8719	0	8808	8808
kroE100	100	50	9102	9176	-0.0081	9370	9370
lin105	100	52	5848	5954	-0.0181	5532	5532
lin318	318	159	18600	19114	-0.0276	17655	17655
pr107	107	53	18028	18028	0	14839	14839
pr124	124	62	22998	22998	0	21420	21564
pr136	136	68	46890	46909	-0.0004	47911	48960
pr144	144	72	28402	**28059**	0.0121	26296	26296
pr152	152	76	36637	38863	-0.0608	30712	30718
Pr76	76	38	41248	42638	-0.0337	37793	37793
pr226	226	113	38718	39597	-0.0227	33349	33349
Rat195	195	97	1140	1160	-0.0175	1106	1108
rat99	99	49	574	574	0	554	554
rd100	100	50	3168	3168	0	3023	3203
St70	70	35	260	260	0	250	264
Swiss42	42	21	458	**186**	0.5939	132	141
U159	159	79	18401	18401	0	16750	17198
Ulysses16	16	8	1685	1685	0	1329	1329
Ulysses22	22	11	1902	1902	0	1473	1473

Table 3

Computational results of proposed GA on instances with $k = \left\lfloor \frac{3*n}{4} \right\rfloor$

Instance	n	k	Best known TSSP solution	Best found TSSP solution	Gap	OTSSP solution by proposed GA	
						Best	Worst
at280	280	210	2066	2094	-0.0136	1894	1925
bayg29	29	21	999	999	0	929	929
bays29	29	21	1194	1194	0	1090	1090
berlin52	52	39	4174	4174	0	3853	3904
Bier127	127	95	50324	50764	-0.0087	51542	51604
burma14	14	10	1642	**1575**	0.0408	1349	1349
ch130	130	97	3907	4158	-0.0642	4062	4127
ch150	150	112	4499	4720	-0.0491	4480	4480
d198	198	148	9386	**9363**	0.0025	7874	7874
dantzig42	42	31	427	427	0	404	404
eil101	101	75	396	406	-0.0253	398	398
eil51	51	38	289	**287**	0.0069	278	286
eil76	76	57	336	336	0	345	348
fri26	26	19	601	601	0	492	492
gil262	262	196	1672	1695	-0.0138	1671	1671
gr137	137	102	43912	48623	-0.1073	44147	44286
gr17	17	12	951	951	0	640	640
gr21	21	15	1501	1501	0	1276	1276
gr24	24	18	844	844	0	763	763
gr48	48	36	3104	3104	0	3135	3135
gr96	96	72	31437	**31095**	0.0109	29257	30083
gr202	202	151	21563	21954	-0.0181	22000	22528
gr229	229	171	69201	**67848**	0.0196	65832	66183
hk48	48	36	7278	7278	0	6937	6937
kroA100	100	75	14492	14492	0	13982	14229
kroA150	150	112	18210	18295	-0.0047	17787	17953
kroA200	200	150	20723	**20135**	0.0284	20705	20855
kroB100	100	75	14744	14744	0	14648	14787
kroB150	150	112	17501	**17349**	0.0087	17090	17193
kroB200	200	150	20508	21266	-0.0370	20553	20851
kroC100	100	75	14067	14067	0	14295	14419
kroD100	100	75	14171	14171	0	13884	14120
kroE100	100	75	14640	14640	0	15347	15347
lin105	105	78	9034	**8999**	0.0039	8412	8412
lin318	318	238	29829	29829	0	27963	28051
pr107	107	80	36468	37605	-0.0312	29684	29684
pr124	124	93	39174	39174	0	36977	36977
pr136	136	102	69690	70790	-0.0158	67879	69818
pr144	144	108	41452	41703	-0.0061	40274	40832
pr152	152	114	57431	**52393**	0.0877	46434	46635
Pr76	76	57	64142	64918	-0.0121	62262	62262
pr226	226	169	47516	49198	-0.0354	49022	49588
rat195	195	146	1753	**1713**	0.0228	1669	1670
rat99	99	74	861	870	-0.0105	868	868
rd100	100	75	5094	5175	-0.0159	5476	5476
St70	70	52	428	428	0	437	446
Swiss42	42	31	760	**333**	0.5618	281	281
U159	159	119	27413	27612	-0.0073	27342	27348
Ulysses16	16	12	3183	3183	0	2704	2704
Ulysses22	22	16	2941	2968	-0.0092	2618	2618

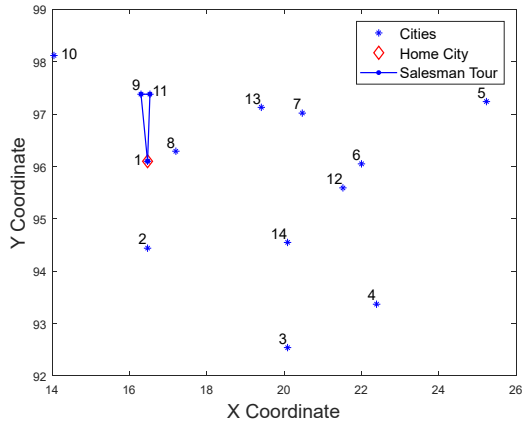
In Table 1, overall, 50 test instances were tested using the proposed GA with $k = \left\lfloor \frac{n}{4} \right\rfloor$. To measure the performance of the proposed GA, the best-found TSSP solutions are compared with the best-known TSSP solutions available in the literature. It

is seen that the number of test instances for which the proposed GA produced solutions coincide with the best-known solutions is 35 out of 50. The number of test instances for which the proposed GA provided solutions worse than the best known are 7 out of 50. Whereas, the number of test instances for which the best-known results improved is 8 out of 50. It is also seen that the negative *Gap* values vary from -0.0469 to -0.006. Whereas positive *Gap* values are ranging from 0.0005 to 0.4792. Furthermore, the best and worst found OTSSP solutions for all the 50 instances generated by the proposed GA are also reported. Similarly, in Table 2, the same test instances were used as used in Table 1 but with $k = \left\lfloor \frac{n}{2} \right\rfloor$. From Table 2, it is evident that the number of test instances for which the proposed GA provided solutions coincides with the best-known solutions is 31 out of 55. The number of test instances for which the produced solutions worse than the best known are 15 out of 55. Whereas, the number of test instances for which the best-known results improved is 4 out of 55. It is also seen that the negative *Gap* values vary from -0.0943 to -0.0004, whereas positive *Gap* values range from 0.0121 to 0.5939. Besides, the best and worst found OTSSP solutions for all these 50 test instances are presented. Finally, in Table 3, the same test instances were used but with $k = \left\lfloor \frac{3*n}{4} \right\rfloor$.

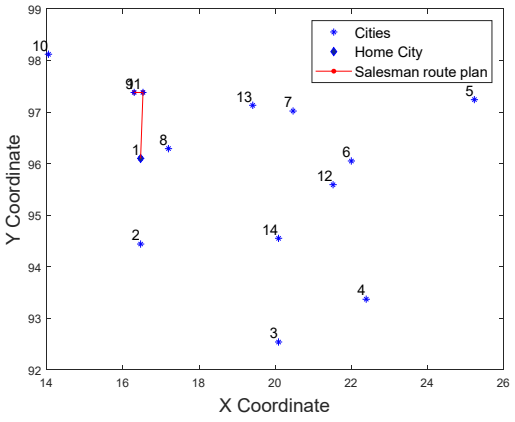
From Table 3, It is observed that the number of test instances for which the obtained results coincides and worse than the best-known results are 20 and 19 out of 50, respectively. The number of test instances for which the best-known results improved is 11 out of 50. It is also seen that the negative *Gap* values vary from -0.1073 % to -0.0047, whereas positive *Gap* values range from 0. 0025 to 0. 5618. Further, the best and worst found OTSSP solutions for these entire 50 test instances tested on proposed GA are also presented. From overall computational results, it is seen that the proposed GA is having a great capability in providing the best TSSP solutions. With this capability, the proposed GA will certainly provide the best solutions for OTSSP in a limited time. Note that 23 out of 150 test cases has improved solutions, which are reported in boldface. These new solutions will be useful as a reference to future comparative studies. Although the structure of TSSP and OTSSP are almost similar with simple relaxation, the optimal solution for TSSP may become the worse solution for OTSSP after removal of edge from the last city to home city. Similarly, the optimal solution for OTSSP may results worse solution for TSSP after including an edge from the last city of the route plan to home city. To show the variation in route plans and its traversal distances as per the structure of the model (TSSP and OTSSP) and different k values, a simple test instance namely *burma14* with 14 cities has been considered. Fig. 9 demonstrates these plots. In Fig. 9, plots (a, b and c) represent the TSSP solutions, whereas plots (d, e, and f) denote the OTSSP solutions generated by the proposed GA with distinct k values (3, 7, 10). In all the plots of this figure, the cities are represented with star symbols labelled with respective city numbers, the home city where the salesman starts and ends his tour is shown with a diamond symbol. This figure clearly shows that the difference in route plans and its traversal distance for the TSSP and OTSSP as per the k value.

5. Conclusions

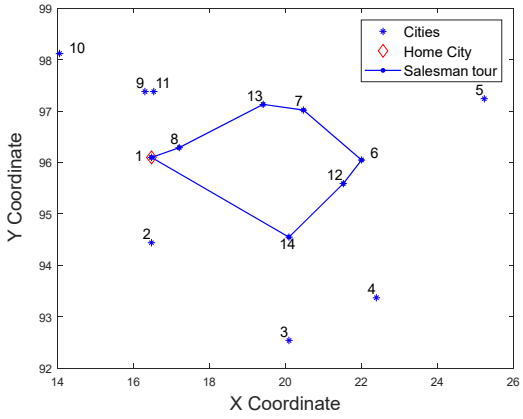
In this study, we have developed a hybrid algorithm that comprises the nearest neighbor technique and crossover free genetic algorithm with complex mutation operators for the OTSSP. The developed algorithm effectively deals with both the features of OTSSP namely, subset selection and permutation. To best of the author's knowledge, this is the first hybrid GA for the OTSSP. As there are no existing benchmark instances for OTSSP, several OTSSP test instances are generated from the TSPLIB to assess the effectiveness of the algorithm proposed. The computational results demonstrate that the proposed algorithm is having great potential in achieving the best results for the OTSSP. However, the physical structure of TSSP and OTSSP models looks similar with simple relaxation (removal of an edge from the last city to home city), but these models are independent to one another. This means that the optimal solution of TSSP may not provide the optimal solution for OTSSP by just deleting an edge from the last city of the route plan to the home city. As the proposed approach is the first GA for OTSSP, this algorithm will be served as the reference approach for measuring the performance of forthcoming heuristic, metaheuristic and hybrid algorithms. Solution techniques analogous to our algorithm can be designed for other models by incorporating effective strategies as per the features of the model. In future, we aim to develop hybrid genetic algorithms for distinct variants of the TSP.



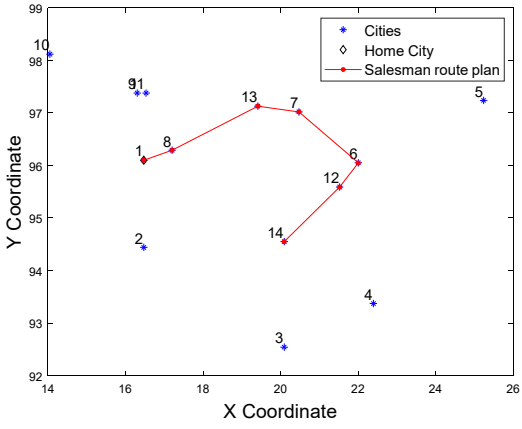
(a) $k=3$ & distance $=280$



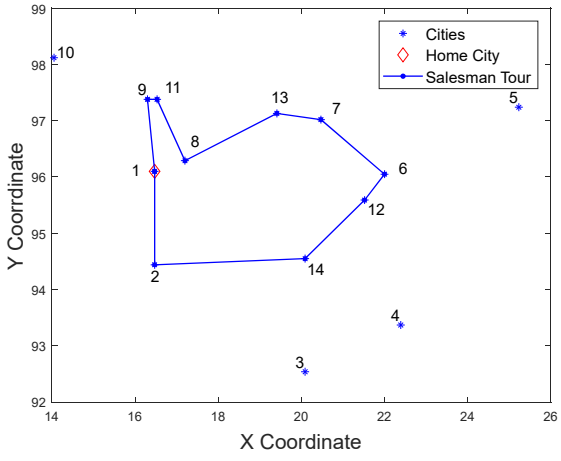
(d) $k=3$ & distance $=151$



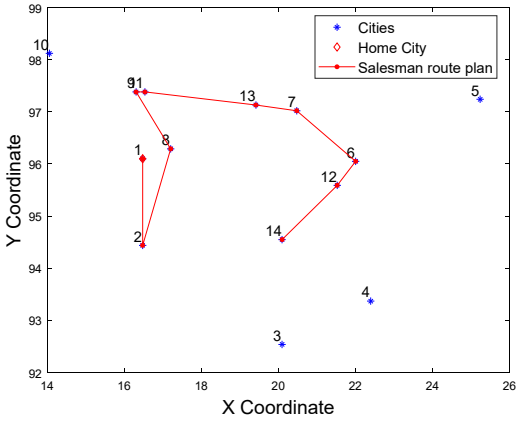
(b) $k=7$ & distance $=1236$



(e) $k=7$ & distance $=842$



(c) $k=10$ & distance $=1575$



(f) $k=10$ & distance $=1349$

Fig.9. Best found TSSP (a, b, c) and OTSSP (d, e, f) solutions by proposed GA on burma14 with distinct k values.

References

Ausiello, G., Bonifaci, V., Leonardi, S., Marchetti-Spaccamela, A., & Gonzalez, T. F. (2018). Prize Collecting Traveling Salesman and Related Problems.

Bahaabadi, M.R., Mohaymany, A.S., & Babaei, M. (2012). An Efficient crossover operator for travelling salesman problem. *International Journal of Optimization in Civil Engineering 2*(4), 607–619.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, *19*(6), 621-636.

Chieng, H. H., & Wahid, N. (2014). A performance comparison of genetic algorithm's mutation operators in n-cities open loop travelling salesman problem. In *Recent Advances on Soft Computing and Data Mining* (pp. 89-97). Springer, Cham.

Gensch, D. H. (1978). An industrial application of the traveling salesman's subtour problem. *AIIE Transactions*, *10*(4), 362-370.

Giardini, G., & Kalmar-Nagy, T. (2011). Genetic algorithm for combinatorial path planning: the subtour problem. *Mathematical Problems in Engineering*, *2011*.

Goldenberg, D. E. (1989). Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Newyork.

Hussain, A., Muhammad, Y. S., Nauman Sajid, M., Hussain, I., Mohamd Shoukry, A., & Gani, S. (2017). Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Computational intelligence and neuroscience*, *2017*.

Ibaraki, T. (1973). Algorithms for obtaining shortest paths visiting specified nodes. *Siam Review*, *15*(2), 309-317.

Laporte, G., Mercure, H., & Norbert, Y. (1984). Optimal tour planning with specified nodes. *RAIRO-Operations Research-Recherche Opérationnelle*, *18*(3), 203-210.

Liu, C., & Kroll, A. (2016). Performance impact of mutation operators of a subpopulation-based genetic algorithm for multi-robot task allocation problems. *SpringerPlus*, *5*(1), 1361.

Maredia, A., & Pepper, R. (2010). History, Analysis, and Implementation of Traveling Salesman Problem (TSP) and Related Problems. *Department of Computer and Mathematical Sciences, University of Houston-Downtown*.

Matai, R., Singh, S. P., & Mittal, M. L. (2010). Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, *1*.

Mittenthal, J., & Noon, C. E. (1992). An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint. *Journal of the Operational Research Society*, *43*(3), 277-283.

Pandiri, V., & Singh, A. (2020). Two multi-start heuristics for the k-traveling salesman problem. *OPSEARCH*, *57*(4), 1164-1204.

Saksena, J. P., & Kumar, S. (1966). The routing problem with "K" specified nodes. *Operations Research*, *14*(5), 909-913.

Stetsyuk, P. I. (2016). Problem statements for k-node shortest path and k-node shortest cycle in a complete graph. *Cybernetics and Systems Analysis*, *52*(1), 71-75.

Venkatesh, P., Srivastava, G., & Singh, A. (2018). A general variable neighborhood search algorithm for the k-traveling salesman problem. *Procedia computer science*, *143*, 189-196.

Verweij, B., & Aardal, K. (2003). The merchant subtour problem. *Mathematical programming*, *94*(2-3), 295-322.

Westerlund, A., Göthe-Lundgren, M., & Larsson, T. (2006). A stabilized column generation scheme for the traveling salesman subtour problem. *Discrete Applied Mathematics*, *154*(15), 2212-2238.