Evolutionary Algorithms for the Traveling Salesman with Multiple Passengers and High Occupancy Problem

Ranmsés Emanuel Martins Bastos,
Marco César Goldbarg,
Elizabeth Ferreira Gouvêa Goldbarg
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Natal, Brazil
ranmses@ufrn.edu.br, gold@dimap.ufrn.br, beth@dimap.ufrn.br

Matheus da Silva Menezes
Departamento de Ciências Exatas e Naturais
Universidade Federal Rural do Semi-Árido
Mossoró, Brazil
matheus@ufersa.edu.br

***Abstract*—As a generalization of the Travelling Salesman Problem, the Travelling Salesman with Multiple Passengers and High Occupancy Problem considers real-world aspects with environmental implications. This paper presents a set of features concerning the study of this new optimization problem, including a mixed-integer programming model and its linearization, a group of artificial instances, and four heuristics based on the Genetic, Memetic, and Transgenetic metaheuristics. The instances are submitted to Gurobi solver to establish a benchmark. Heuristics' parameters are tuned using the Irace tool and later compared through a computational experiment. A statistical analysis based on Friedman tests pointed to a superior performance of TGALS.**

Keywords—Ridesharing, vehicle routing, metaheuristics.

I. Introduction

The Travelling Salesman with Multiple Passengers and High Occupancy Problem (TSMPHOP) is an extension of the Traveling Salesman Problem (TSP). The latter is one of the most traditional and well-known mathematical programming problems [1]. TSP's objective is to find the lowest cost route that travels across a given set of cities. The TSMPHOP leverages this scenario and adds some real-world aspects to it: cities are visited by the salesman's vehicle, which has seats available; the salesman offers rides to passengers along the TSP route to share expenses; roads connect cities, and some of them are toll roads; tolls are said to be of the High-Occupancy type since vehicles are exempt from paying the fare if they have no seats available; when charged, toll's expenses are not shared: they are entirely paid by the salesman; lastly: shared costs are equally divided between the salesman and all riding passengers on their respective paths. TSMPHOP's objective is to find the Hamiltonian cycle with the lowest cost, which is defined by the sum of expenses paid by the salesman along the route. We also consider that there are no passengers available for pickup at the salesman origin since routes always start at it. Besides the aforementioned aspects, there are also passenger-related restrictions. The first one is that each of them has specific pickup and drop-off cities. The second is that the share of expenses due to a passenger is subject to his budget limit. Those restrictions are mandatory; all passengers' transportation must be done in conditions that respect them.

It should be noted that the collaborative bias in the problem is remarkable since the salesman has the incentive to share his means and thus decrease the overall cost by providing transport to third party individuals. To present the definition of TSMPHOP, Section II contains the mathematical formulation and the linearization of the nonlinear expressions involved. The linearized model's implementation comprises the exact solving method. Section III introduces four algorithms designed to heuristically solve the problem, which are based on the Genetic, Memetic, and Transgenetic metaheuristics. To demonstrate the generation of TSMPHOP's artificial instances and the test methodology adopted in this paper, Section IV follows. Computational experiments' results are discussed in Section V, while Section VI conveys conclusions of the work.

II. TSMPHOP

Since the TSP is a particular case of the TSMPHOP, the latter belongs to NP-hard [2] [3]. TSMPHOP is a model that includes routing and ride-sharing problems. It belongs to the class of green transportation problems since it can produce results that benefit environmental preservation.

The TSMPHOP consists of finding a Hamiltonian cycle in a complete and nondirectional graph G and the list of passengers whose transport requests are satisfied throughout the route. Restrictions shall ensure the conditions required by the passengers, such as pickup and drop-off points and budget limits, are successfully met. The vehicle capacity also has to be respected. The vehicle has C seats available to passengers. The expenses for going from a city to another are composed by the inherent edge cost, such as fuel and vehicle maintenance, and the toll cost, which may or may not exist. When there is a toll occurrence in a given edge, such an edge is from now on called a *hov* edge (from *high-occupancy*

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

vehicle lane). The toll fare is closely related to the vehicle occupation: it is only charged to the salesman if it is not full in its maximum occupation C. If all seats are occupied, the vehicle is exempt, and the toll fare is zero.

The object is to minimize (1), where d_{ij} denotes the cost of edge (i,j); x_{ij} is a binary variable equal to 1 if the edge (i,j) belongs to the Hamiltonian cycle and 0 otherwise; φ_{ij} is a binary variable with value 0 whenever the vehicle passes by (i,j) with maximum occupancy and value 1 otherwise; w_{ij} is the cost of the toll associated with the edge (i,j), which is nonzero for *hov* edges and zero otherwise. L is the set of people demanding transportation; v_{ij}^{l} is a binary variable equal to 1 if passenger l is in the vehicle in edge (i,j) and 0 otherwise. P_l is the pickup point of passenger l. Q_l is the drop-off point of passenger l. The first term in the objective function accomplishes the division of expenses between the salesman and the car occupants. The second term ensures maximum incentive for the salesman to share its means since the toll fare is not shared with the passengers but can be zero if all seats are occupied.

$$\min \sum_{i,j \in N} \frac{d_{ij}x_{ij}}{\sum_{k\in L} v_{ij}^{k}+1} + \varphi_{ij}w_{ij}x_{ij} \quad (1)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in N \quad (2)$$

$$\sum_{j=1}^{n} x_{ji} = 1 \qquad \forall i \in N \quad (3)$$

$$u_1 = 1 \quad (4)$$

$$u_i - u_j + 1 \le (n-1)(1-x_{ij}) \qquad \forall i,j \in N\backslash\{1\} \quad (5)$$

$$\sum_{l=1}^{L} v_{ij}^{l} - Cx_{ij} \le 0 \qquad i \ne j \quad \forall i,j \in N \quad (6)$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\ne i}}^{n} \frac{v_{ij}^{l}d_{ij}}{\sum_{k=1}^{L} v_{ij}^{k}+1} - t_l \le 0 \qquad \forall l \in L \quad (7)$$

$$\varphi_{ij} = 1 - \left\lfloor \frac{\sum_{l=1}^{L} v_{ij}^{l}+1}{C+1} \right\rfloor \qquad i \ne j \quad \forall i,j \in N \quad (8)$$

$$\sum_{\substack{j=1\\ j\ne i}}^{n} v_{ij}^{l} - \sum_{\substack{j=1\\ j\ne i}}^{n} v_{ji}^{l} = 0 \qquad \begin{matrix} i \ne P_l, Q_l \quad \forall i \in N \\ \forall l \in L \end{matrix} \quad (9)$$

$$\sum_{\substack{i=1\\ i\ne P_l}}^{n} v_{iP_l}^{l} + \sum_{\substack{i=1\\ i\ne Q_l}}^{n} v_{Q_l i}^{l} = 0 \qquad \forall l \in L \quad (10)$$

$$\sum_{i=2}^{n} v_{1i}^{l} = 0 \qquad \forall l \in L \quad (11)$$

$$v_{ij}^{l} \le x_{ij} \qquad i \ne j \quad \forall i,j \in N \quad \forall l \in L \quad (12)$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in N \quad (13)$$

$$u_i \in N\backslash\{1\} \qquad \forall i \in N\backslash\{1\} \quad (14)$$

$$v_{ij}^{l} \in \{0,1\} \qquad \forall i,j \in N \quad \forall l \in L \quad (15)$$

$$\varphi_{ij} \in \{0,1\} \qquad \forall i,j \in N \quad (16)$$

Equations (2) and (3) ensure that all cities are visited once. Expressions (4) and (5) prevent subcycles, which came from the MTZ formulation for the TSP [4]. Constraint (5) ensures that each city has a unique associated visit order. Constraint (6) requires that at most C passengers are boarded on any edge of the route. Constraint (7) ensures that each passenger pays, at most, the maximum rate agreed, t_l. Equation (8) models the toll exemption, making the toll rate w_{ij} to be charged to the salesman whenever there are unoccupied seats in the vehicle and edge (i,j) is part of the cycle. Equation (9) ensures that embarked passengers are always disembarked. Constraint (10) ensures that passenger l is not in the vehicle on an edge that arrives at P_l or leaves Q_l. Equation (11) ensures that no passenger that arrives at the salesman origin city continues in the route. Constraint (12) ensures that no passenger is in the vehicle on an edge that is not in the cycle. Constraints (13) to (16) define the decision variables' domains.

A. Linearization

Expressions (1), (7), and (8) are nonlinear. The linearization of such is based on the following mathematical formulas.

i. Product between binary variables a and b [5].

$$c = ab \qquad a,b,c \in \{0,1\} \quad (17)$$
$$c \le a \quad (18)$$
$$c \le b \quad (19)$$
$$c \ge a + b - 1 \quad (20)$$

ii. Product between binary variable a and continuous positive variable b with constant upper limit B [6].

$$c = ab \qquad a \in \{0,1\} \quad (21)$$
$$c \le aB \qquad 0 \le b \le B \quad (22)$$
$$c \le b \qquad c \ge 0 \quad (23)$$
$$c \ge b - (1-a)B \quad (24)$$

iii. Floor function mathematical definition [7].

$$a = \lfloor b \rfloor \qquad a \in N \quad (25)$$
$$0 \le b - a \qquad b > 0 \quad (26)$$
$$b - a \le \varepsilon \qquad \varepsilon = 0.999 \quad (27)$$

1) Linearization of (8). Based on the discontinuity shown in (iii), equation (8) can be expressed in a linear way by means of expressions (28) and (29).

$$0 \le \frac{\sum_{l=1}^{L} v_{ij}^{l}+1}{C+1} - \left(1-\varphi_{ij}\right) \qquad \begin{matrix} i \ne j \\ \forall i,j \in N \end{matrix} \quad (28)$$

$$\frac{\sum_{l=1}^{L} v_{ij}^{l}+1}{C+1} - \left(1-\varphi_{ij}\right) \le 0.999 \quad (29)$$

2) Linearization of (7). The occupancy of the vehicle on edge (i,j), given by $\left(\sum_{k=1}^{L} v_{ij}^{k}+1\right)$, can only assume a single integer value in the interval $[1, C+1]$. It is possible to leverage this fact to decompose the inverse of the occupancy expression into a sum. By introducing the binary variable λ_{ij}^{m}, expressions (30) and (31) demonstrate how it can be done.

$$\frac{1}{\sum_{k=1}^{L} v_{ij}^{k}+1} = \sum_{m=1}^{C+1} \lambda_{ij}^{m}\left(\frac{1}{m}\right) \qquad \begin{matrix} \lambda_{ij}^{m} \in \{0,1\} \\ \forall i,j \in N \\ \forall m \in 1 \dots C+1 \end{matrix} \quad (30)$$

$$\sum_{m=1}^{C+1} \lambda_{ij}^{m} = 1 \qquad \begin{matrix} i \ne j \\ \forall i,j \in N \end{matrix} \quad (31)$$

Equation (30) is nonlinear; therefore, we need to find a way to express it in a linear form. We start by rearranging it to obtain a sum of products indicated by the equation (32).

$$\sum_{m=1}^{C+1}\left(\lambda_{ij}^{m}\left(\sum_{k=1}^{L} v_{ij}^{k}+1\right)\left(\frac{1}{m}\right)\right)=1 \qquad \begin{matrix} i\neq j \\ \forall i,j\in N \end{matrix} \qquad (32)$$

To linearize the new products in (32), the variable σ_{ij}^{m} is introduced in equation (33).

$$\sigma_{ij}^{m}=\lambda_{ij}^{m}\left(\sum_{k=1}^{L} v_{ij}^{k}+1\right) \qquad \begin{matrix} \forall i,j\in N \\ \forall m\in 1\dots C+1 \end{matrix} \qquad (33)$$

Equation (32) is rewritten as (34).

$$\sum_{m=1}^{C+1}\sigma_{ij}^{m}\left(\frac{1}{m}\right)=1 \qquad \begin{matrix} i\neq j \\ \forall i,j\in N \end{matrix} \qquad (34)$$

Applying (ii) and the car maximum occupancy as upper limit $V_{ij}^{k}=\left(\sum_{k=1}^{L} v_{ij}^{k}+1\right)\leq C+1$, the products of (33) are now linear as displayed by the expressions (35) to (38):

$$\sigma_{ij}^{m}\leq\lambda_{ij}^{m}(C+1) \qquad \forall i,j\in N \qquad (35)$$

$$\sigma_{ij}^{m}\leq V_{ij}^{k} \qquad \forall m\in 1\dots C+1 \qquad (36)$$

$$\sigma_{ij}^{m}\geq V_{ij}^{k}-(1-\lambda_{ij}^{m})(C+1) \qquad (37)$$

$$\sigma_{ij}^{m}\geq 0 \qquad (38)$$

Combining $\left(\sum_{k=1}^{L} v_{ij}^{k}+1\right)\geq 1$ with (33), it is possible to replace (38) by (39).

$$\sigma_{ij}^{m}\geq\lambda_{ij}^{m} \qquad \begin{matrix} \forall i,j\in N \\ \forall m\in 1\dots C+1 \end{matrix} \qquad (39)$$

So far, the term $\frac{1}{\sum_{k=1}^{L} v_{ij}^{k}+1}$ can be correctly expressed by $\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)$. For the complete linearization of (7), it is necessary to treat the slightly different expression as follows: $\frac{v_{ij}^{l}}{\sum_{k=1}^{L} v_{ij}^{k}+1}=v_{ij}^{l}\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right)$. Variable γ_{ij}^{l} is introduced in equation (40) for this purpose.

$$\gamma_{ij}^{l}=v_{ij}^{l}\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right) \qquad \begin{matrix} \forall i,j\in N \\ \forall l\in L \end{matrix} \qquad (40)$$

Observing that (30) implies $\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\leq 1$, it is possible to apply (ii) to (40) and obtain the linear expressions (41) to (44).

$$\gamma_{ij}^{l}\leq v_{ij}^{l} \qquad \forall i,j\in N \qquad (41)$$

$$\gamma_{ij}^{l}\leq\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right) \qquad \forall l\in L \qquad (42)$$

$$\gamma_{ij}^{l}\geq\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right)-(1-v_{ij}^{l}) \qquad (43)$$

$$\gamma_{ij}^{l}\geq 0 \qquad (44)$$

Equation (30) also implies $\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\geq\frac{1}{C+1}$, which provides a way to replace (44) by (45).

$$\gamma_{ij}^{l}\geq\frac{v_{ij}^{l}}{C+1} \qquad \begin{matrix} \forall i,j\in N \\ \forall l\in L \end{matrix} \qquad (45)$$

Finally, (7) can be expressed by (46).

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\gamma_{ij}^{l}d_{ij}-t_{l}\leq 0 \qquad \forall l\in L \qquad (46)$$

3) Linearization of (1). First, the variable α_{ij} is introduced by equation (47). It is possible to obtain its linearization by applying the same approach used for γ_{ij}^{l}, which gives expressions (48) to (51).

$$\alpha_{ij}=\frac{x_{ij}}{\sum_{k=1}^{L} v_{ij}^{k}+1}=x_{ij}\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right) \qquad \forall i,j\in N \qquad (47)$$

$$\alpha_{ij}\leq x_{ij} \qquad (48)$$

$$\alpha_{ij}\leq\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right) \qquad (49)$$

$$\alpha_{ij}\geq\left(\sum_{m=1}^{C+1}\lambda_{ij}^{m}\left(\frac{1}{m}\right)\right)-(1-x_{ij}) \qquad (50)$$

$$\alpha_{ij}\geq 0 \qquad (51)$$

Employing an analogous argument to what was used to obtain (45), restriction (51) can be replaced by (52):

$$\alpha_{ij}\geq\frac{x_{ij}}{C+1} \qquad \forall i,j\in N \qquad (52)$$

Second, equation (53) defines the binary variable β_{ij}. By applying (i) to linearize it, expressions (54) to (56) are obtained.

$$\beta_{ij}=x_{ij}\varphi_{ij} \qquad \forall i,j\in N \qquad (53)$$

$$\beta_{ij}\leq x_{ij} \qquad (54)$$

$$\beta_{ij}\leq\varphi_{ij} \qquad (55)$$

$$\beta_{ij}\geq x_{ij}+\varphi_{ij}-1 \qquad (56)$$

Ultimately, the objective function (1) is now linear:

$$\min\sum_{i,j\in N}\alpha_{ij}d_{ij}+\beta_{ij}w_{ij} \qquad (57)$$

In short, the nonlinear constraints were replaced as follows: equation (8) by expressions (28), (29); expression (7) by expressions (31), (34) to (37), (39), (41), (42), (43), (45), (46); and expression (1) by (48), (49), (50), (52), (54) to (57).

III. Algorithms

This section comprises the methods developed to obtain the heuristic solutions of TSMPHOP, including standardized auxiliary procedures and metaheuristic algorithms.

A. Auxiliary Procedures

The procedures described in this section deal with aspects related to neighborhoods and methods used to manipulate solutions, being common to all algorithms.

1) Neighborhood Structures – Operators. Fig. 1 illustrates the neighborhood structures utilized in the algorithms presented in this research. The salesman origin, city 1, is omitted for simplicity, but it is considered to be at the beginning and the end of each route in the examples.

The *change* operator exchanges the positions of two cities in the route. Let $S=(c_1,\dots,c_{n-1})$ be a permutation that represents a route and $S'=(c_i,\dots,c_j)$ a substring of S, $1<i<j\leq n-1$. The *insertion* operator replaces the substring S' by $S''=(c_{i+1},\dots,c_j,c_i)$. The *inversion* operator reverts the order of S''. These operators change the order of the cities disregarding any quality information of the problem.

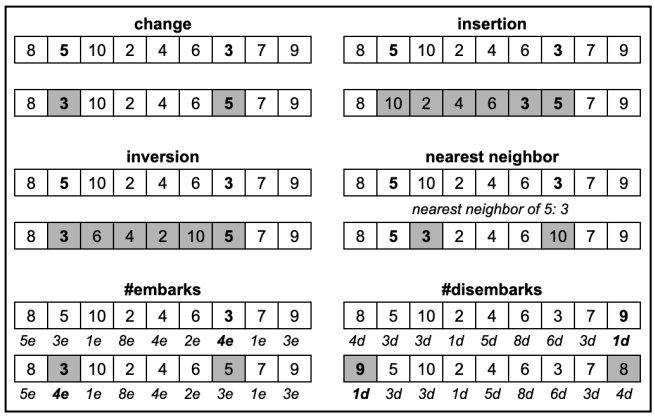

Fig. 1. Neighborhood operators.

The *change* operator exchanges the positions of two cities in the route. Let $S = (c_1, \dots, c_{n-1})$ be a permutation that represents a route and $S' = (c_i, \dots, c_j)$ a substring of S, $1 < i < j \leq n-1$. The *insertion* operator replaces the substring S' by $S'' = (c_{i+1}, \dots, c_j, c_i)$. The *inversion* operator reverts the order of S''. These operators change the order of the cities of a route disregarding any quality information of the problem.

The other three operators seek to reduce the total cost of the route. Let s_i be a city in S, $1 < i < n-1$, the *nearest neighbor* operator computes the nearest city from s_i, say s_j, and swaps s_{i+1} and s_j. The *#embarks* operator aims to place cities that are the pickup point of the highest number of passengers at the beginning of the route. First, we choose a city at random and check the number of passengers available for pickup on it. In the example, this is the city 3, which has four embarking passengers (indicated by 4e). Then, starting from the route's beginning, we search for the first occurrence of a city with less embarking passengers. This is city 5 in the example, which has three embarking passengers. Once both cities are determined, we switch their positions in the route.

Similarly, the *#disembarks* operator aims to place cities that are the drop-off point of the highest number of passengers at the end of the route. The only difference here is that we look for a second city that has more disembarking passengers than the first one. In the example, the first one is city 9, which has one disembarking passenger (symbolized by 1d), and the second is city 8, which has four disembarking passengers.

2) Passenger Loading procedure. The standard routine to assign passengers to a defined route, named PL, is semi-greedy. Since there are no passengers available for pickup in the first city (the salesman's origin), we analyze the route going city by city starting from the second one and proceeding until the end or until we find a stopping condition. In each city, we try to load the most passengers into the vehicle whenever there are empty seats. To optimize the search for passengers, we only consider *viable* passengers for this specific route, i.e., those who have the pickup city being visited before the drop-off city. This pre-selection mechanism prevents the undesirable condition of allowing passengers to board without being capable of providing transportation until the destination.

To define the order on which passengers will be first selected for embark, we randomly chose one of three mechanisms beforehand: m_1 – every passenger has an equal probability; m_2 – the higher the passenger's maximum fare, the higher the probability; or m_3 – the larger the number of cities between passenger's pickup and drop-off points, the higher the probability. When we have no free seats or there are no more passengers to load in a given city, we verify the budget limit restraint for each passenger on board. If a passenger paying more than its maximum fare, said passenger is marked as prohibited, and we have encountered a stopping condition. In that case, we restart the procedure from the beginning (second city), but now excluding any marked passenger. We accept the passenger loading scheme as successful when we reach the route's end satisfying all passenger-related restrictions.

Algorithm 1 shows the pseudocode of the PL, where L^* is the set of prohibited passengers; S, the route comprised by cities $c_1, \dots, c_n$; L_v, the set of *viable* passengers for route S; m, the mechanism that defines the order of passengers we analyze for embarkation; and $validscheme$, a boolean variable. The outer loop executes while there is a budget nonconformity. In every iteration of the outer loop, L_v excludes marked passengers (line 5) and sets the control variable as $true$. The algorithm assigns passengers to the vehicle in the inner loop (line 6) on a per city basis. First, the procedure removes passengers that drop off at the *i*-th city (lines 7-8). If there are empty seats, passengers are picked up in the order defined by $first(L_v, m_i, c_i)$ (lines 9-10). Routine $budgetcheck()$ checks budget violations for passengers in the vehicle, updating l^*. Lines 12-15 check if the loop continues to the next city or if it stops.

Algorithm 1 Passenger Loading procedure (PL)

1 $L^* \leftarrow \emptyset$; $S \leftarrow (c_1, \dots, c_n)$; $L_v \leftarrow viable(S)$
2 $m \leftarrow random(m_1, m_2, m_3)$; $validscheme \leftarrow false$
3 **While** $validscheme = false$ **do**
4 $i \leftarrow 2$; $l^* \leftarrow null$
5 $L_v \leftarrow L_v \backslash L^*$; $validscheme \leftarrow true$
6 **While** $i \leq n$ **do**
7 **If** (there are passengers to drop off at c_i) **then**
8 $disembark()$
9 **For** $l \in first(L_v, m, c_i)$ **do**
10 **If** (there are empty seats) **then** $embark(l)$
11 $l^* \leftarrow budgetcheck()$
12 **If** l^* is $null$ **then** $i \leftarrow i + 1$
13 **Else**
14 $i \leftarrow n + 1$; $L^* \leftarrow L^* \cup \{l^*\}$
15 $discardscheme()$; $validscheme \leftarrow false$

3) Optimal Passenger Loading. In order to improve solution quality, we utilize an exact procedure called $eOPT$ in the heuristics. This procedure is a reduced version of the linearized model. Both of them are solved using Gurobi [8]. Since we have a defined route in this scenario, the model's variables x_{ij} and u_i become constants. Then, all related equations and expressions are greatly simplified or even cease to exist. We also only consider the *viable* passengers for the route (as defined in PL). Hence, we have a routine specially designed to, starting from a previously defined route and a set of passengers, mathematically determine the optimal passenger loading scheme. Since this method is computationally costlier than PL, only a small number of the generated individuals are submitted to it. In all of the

developed heuristics, $eOPT$ receives the current generation best individual p_{min} and the current best global solution s^*. The routine takes s^* as an upper bound, creates a new solution with p_{min} route loaded with the optimal scheme, and updates s^* if the new solution is better.

4) Initial Population. We produce the starting set of solutions by running the $genPop$ procedure, described as follows. The first element is a route generated by the Lin-Kernighan heuristic [9] applied to a random cycle. We obtain the next elements by submitting the original one to random movements until we have the desired population size. Last, we apply PL to each generated route to load them.

B. Genetic and Memetic Algorithms

The first algorithm developed is a Genetic Algorithm (GA). It has the following input parameters: $tPop$ – the number of individuals in the population; $gMax$ – the maximum number of generations; $rElite$ – the percentage of elite solutions; pCr – the crossover probability; and $pMut$ – the mutation probability.

We implemented the following mechanisms: Selection, executed by binary tournament, Crossover, Mutation, and Elitism. Crossover simulates the reproductive process according to a pre-defined probability. It ensures that each of the parent solutions provides genetic material to the generation of two child solutions, being chosen as the offspring the best among them. To implement the Crossover mechanism, we applied the so-called ordinal representation presented by [9], which is a special representation for maintaining the feasibility of a population composed of tours. Once we have the route of both parent solutions converted to this format, we execute the process by choosing the routes' half as the crossover point. We obtain two new routes by interchanging their parents' halves. Next, we decode the new routes back to the original representation and assign passengers to them by applying PL, creating two child solutions as a result. Last, we select the best between them. Mutation happens through the application of one randomly chosen operator from Fig.1 to the offspring according to a pre-defined probability. Elitism ensures that a percentage of the best solutions of a given generation always remain in the next one.

Algorithm 2 illustrates GA's pseudocode. The initial population *P* is generated on line 1, and s^* is initialized in line 2 with the best solution so far, p_{min}. Lines 3-13 denote the outer loop, which controls the generations' number. In line 4, the next population P' is initialized with a percentage of the best individuals from the current population P. Lines 5-10 contains the inner loop, which creates new individuals in P' until we get to the desired population size. Lines 6-7 select a pair of solutions p_1 and p_2 to serve as parents in line 8, which implements the crossover mechanism to obtain an offspring p_3. Line 9 executes the mutation process, lastly creating p_4, which is, in line 10, included in P'. In lines 11 and 12, the newly formed population becomes current as a preparation for the start of a new generation. Line 13 tries to improve the quality of the best solution in the current population by optimally loading its route through $eOPT$ and updating the best global solution if needed. Last, line 14 returns the solution s^* as the algorithm's result.

Being very similar to the GA, the Memetic Algorithm (MA) simulates the phenotypic manifestation by replacing the mutation process with a local search procedure named $opLS$. First, we randomly select one of the operators from Fig.1. Next, instead of stopping the search for the second city at the first occurrence (as noted in section III, subsection A.1), we let the search cover all cities, thus obtaining a local search. We then accept the best solution generated as the search's result. Besides $pMut$, all of GA's parameters are also present in MA. The pseudocode of MA is the same of Algorithm 2 except for line 9, which is replaced by $p_4 \leftarrow opLS(p_3)$.

Algorithm 2 Genetic Algorithm (GA)

```
1   P ← genPop(tPop)
2   s* ← p_min
3   For i ← 1 to gMax do
4       P' ← elitism(P, rElite)
5       While |P'| < tPop do
6           p_1 ← selection(P)
7           p_2 ← selection(P)
8           p_3 ← crossover(p_1, p_2, pCr)
9           p_4 ← mutation(p_3, pMut)
10          P' ← P' ∪ {p_4}
11      P ← P'
12      P' ← ∅
13      eOPT(s*, p_min)
14  Return s*
```

C. Transgenetic Algorithms

The Transgenetic Algorithm (TGA) is an evolutionary metaheuristic proposed by [11]. It mimics genes horizontal transfer. The strategy is to apply the transfer of genetic material between endosymbionts and host. The TGA operators are plasmids and transposons. They are methods to explore the solution space. There is a solution repository, called GIR (Genetic Information Repository), from where the algorithm gets information (parts of solutions) for the plasmids. The TGA presented in this study is a variation of the algorithm for the Prize-collecting Traveling Car Renter Problem (pCaRS) presented by [12]. The TGA input parameters are $tPop$ – the number of individuals in the population; $tGIR$ – population percentage that defines the number of individuals of GIR; $itMax$ – the maximum number of iterations; and $estMax$ – the maximum number of evolutionary stages. The plasmid is a method to manipulate the individuals of the population. It consists of inserting information, i.e., fragments of other solutions, in the individuals of *P*. In this study, the information of the plasmid comes from a solution stored in GIR. It consists of a set with the best $tGIR$ solutions of the current population. The procedure may occur in a *partial* or *total* mode, both being equiprobable.

Let p be an individual from P having as route $H = (h_1, \dots, h_{n-1})$. If a plasmid in the *partial* mode manipulates p, we randomly select a donor d from GIR, a fragment size t in the range [2; 0.4N], and the fragment's starting position $k, 1 < k < n - t$. Suppose d has the route $F = (f_1, \dots, f_{n-1})$. In this scenario, p is our host, d is our endosymbiont, and our fragment $info = (f_{k+1}, \dots, f_{k+t})$. We start by excluding of H all the cities present in $info$, therefore getting $H' = (h'_1, \dots, h'_{n-t-1})$ where $h'_i \neq f_j$, $1 \leq i < n - t$ and $k < j \leq k + t$. Then, we take every possible position for inserting

$info$ as a whole fragment into H'. For example, by inserting $info$ at the second position we would obtain $(h'_1, h'_2, \boldsymbol{f_{k+1}}, \dots, \boldsymbol{f_{k+t}}, h'_3, \dots, h'_{n-t-1})$. Next, each newly formed route is loaded with passengers by applying PL. Last, we select the solution with better quality as the procedure result. For the *total* mode, the only difference is that we consider all possible values for k instead of randomly selecting one just one.

The transposon is also a method to manipulate individuals. However, it does not use information from the environment: it simply reorganizes the genetic material contained in the individual itself. We implemented this process by applying one randomly chosen operator from Fig.1 to the solution at hand. There is no local search in this method: it is a one-movement modification in the solutions' route that is then submitted to PL to receive a passenger loading scheme.

When TGA starts, there is a proclivity towards the $plasmid$ operator intended to enrich the population with quality information. As the algorithm reaches its end, such a trend is gradually reversed in the direction of the $transposon$ method. We achieve that by updating the integer variables $sTrend$ and $rTrend$ at each evolutionary stage. The $sTrend$ value starts small and increases until $itMax * estMax$, while $rTrend$ is a randomly chosen number in the range $[1; itMax * estMax]$.

Algorithm 3 Transgenetic Algorithm (TGA)

```
1   P ← genPop(tPop)
2   s* ← p_min
3   For i ← 1 to itMax do
4       sTrend ← i * estMax
5       For j ← 1 to estMax do
6           GIR ← best(P, tGIR)
7           For p ∈ P do
8               rTrend ← random(1, itMax * estMax)
9               If rTrend > sTrend
10                  info ← selection(GIR)
11                  p ← plasmid(info, p)
12              Else
13                  p ← transposon(p)
14      eOPT(s*, p_min)
15  Return s*
```

Algorithm 3 is described as follows. Lines 1 and 2 initialize the population P and the best global solution s^*. Lines 3-14 comprehends the outer loop that executes the iterations. Line 4 updates the $sTrend$, which is first variable that controls the proclivity towards the TGA operators. Lines 5-13 compose the middle loop, which implements the evolutionary stages. Line 6 updates the GIR repository with the best $tGIR$ solutions of P. Lines 7-13 consist of the inner loop, which alters each individual, hence generating a totally new population. The second variable for adjusting the proclivity, $rTrend$, is updated in line 8 and compared to $sTrend$ in the next line. Lines 9-13 decide according to the trend whether the individual at choice will be modified by $plasmid$ or $transposon$. Similarly to the line 13 of Algorithm 2, line 14 applies $eOPT$ to the best solution of the current population, p_{min}, assigning passengers to it with the optimal loading scheme and updating the best global s^* if necessary. The algorithm concludes its execution in line 14, where the best global s^* is returned.

We have designed the Transgenetic Algorithm with Local Search (TGALS) by modifying TGA to make use of the same phenotypic manifestation process explained in the description of the $opLS$ procedure. The $transposon$ method is replaced by the same local search employed in the MA algorithm, thus obtaining the $transposonLS$ method. The pseudocode of TGALS is the same as displayed in Algorithm 3 except for line 13, which is substituted by $p \leftarrow transposonLS(p)$.`

TGALS' strategy is to enrich the evolutionary process of gens' lateral transmission since the phenotype change also occurs in this scenario. The parameters list of TGALS is the same as of TGA.

IV. Test Methodology

We created an instance generator to provide means to carry out comparative investigations between the proposed algorithms. The source code and generated set are available at http://www.dimap.ufrn.br/lae/en/projects/TSMPHOP.php.

The dataset contains 120 instances with 10, 20, and 30 cities. The values of the edge costs are in the range [150;250]. The percentage of edges for which there is a toll is in the range [15;50]. Toll cost, if nonzero, is a percentage, in the range [70;200], applied to the associated edge cost. Nonzero toll probability is related to the cost of the associated edge as follows: (1) cost in range [100;125]: probability in range [70;90], (2) cost in range [125;150]: probability in range [50;70], (3) cost in range [150;175]: probability in range [30;50], and (4) cost in range [175;250]: zero probability. The lower the cost of edge, the greater the probability of occurrence of nonzero toll. The vehicle capacity can be 3, 4, 5, or 6. The limit for the value a passenger agrees to pay is a percentage of the cost of the minimal spanning tree of the graph. It also relates to vehicle capacity. The percentage range is [10;25] on instances such that the vehicle capacity is 3. The percentage range is [10;20] for the remaining instances. The number of passengers in each city (except for the salesman origin city, which has zero passengers) is randomly chosen from one of three ranges: $[0;C]$, $[0;2C]$, and $[0;3C]$, where C denotes the vehicle capacity.

The type of instance regards the capacity of the vehicle and symmetry. There are eight types of instances: types 01 to 04 denote asymmetric problems with C values of 3,4, 5, and 6, respectively, and types 05 to 08 comprise symmetric instances with those same values for C. We divided the instances into 24 groups based on the number of cities and type. The name of each instance is a string that shows its group (size and type) and identifier. For example, the "10-08_02" string is the name of an instance from group "10-08" (10 cities and type 08) and whose identifier is 02.

For the computational experiment, there is a *training* set, with 24 instances (symmetric and asymmetric instances whose identifier is 05), and a *base* set consisting of the remaining ones (identifiers from 01 to 04).

We tuned the heuristics' parameters on an experiment containing only the instances of the *training* set. We used Irace [13]. It requires a setting named *maxExperiments*, which was set to 10^3, to limit the number of experiments. Table I exhibits the input ranges for Irace and the tuning process results.

We carried out two experiments on the *base* set. We refer to them as *unlimited* and *limited*. In the *unlimited* experiment, the stop criterion was the maximum number of iterations (*gMax* for the GA and MA, and *itMax* for the TGA and TGALS). The stop criterion of the *limited* experiments was 10^4 objective function evaluations for each execution of an algorithm. The purpose of the *unlimited* experiment was to enable comparisons between results produced by the solver and the metaheuristics. We also compared the performances of the algorithms concerning the results of the first experiment. The second experiment aimed at comparing the heuristic algorithms, since all of them received the same resources, i.e., the same number of evaluations, therefore mitigating a possible influence of the maximum number of iterations parameter. In both experiments, there were 30 independent executions of each algorithm for each instance.

TABLE I
PARAMETER TUNING

Algorithm	Parameter	Range	Selected
GA	pCr	[0.80; 0.99]	0.86
GA	$pMut$	[0.01; 0.05]	0.05
GA	$tPop$	[75; 150]	109
GA	$gMax$	[100; 200]	180
GA	$rElite$	[0.05; 0.20]	0.08
MA	pCr	[0.80; 0.99]	0.88
MA	$tPop$	[25; 75]	57
MA	$gMax$	[25; 75]	69
MA	$rElite$	[0.05; 0.20]	0.12
TGA	$tPop$	[5; 15]	14
TGA	$itMax$	[5; 15]	15
TGA	$estMax$	[5; 15]	15
TGA	$tGIR$	[0.20; 0.40]	0.28
TGALS	$tPop$	[5; 15]	12
TGALS	$itMax$	[5; 15]	14
TGALS	$estMax$	[5; 15]	15
TGALS	$tBIG$	[0.20; 0.40]	0.26

We registered the value of the best solution found by each heuristic (for each instance), x_{min}; the number of instances for which $x_{min} \leq x_S$, where x_S denotes the solution obtained by the solver (x_S can be the optimal solution, if the optimizer could solve the problem before reaching 80,000s, or the best found solution when the time has expired); and the average processing time in seconds, T_m, for each heuristic. By comparing x_{min} with x_S, it was possible to find x^*, the best solution found for each instance. We executed the Friedman aligned ranks test, an advanced version of the Friedman test. It is a nonparametric statistical test for verifying differences between more than two samples [14]. We applied this test to the gap G, presented in equation (58), and the processing time *T*, measured in seconds, regarding each instance group.

$$G = \frac{x_{min}}{x^*} - 1 \tag{58}$$

The significance level was 5%. The null hypothesis was that there were no significant differences between the datasets compared. If the statistical test pointed out a significant difference, we proceeded to the Friedman *post-hoc* test with Bergmann and Hommel's correction for pairwise comparisons. If the latter test was significant for an instance group regarding a pair of heuristics, it meant that one of those algorithms behaved significantly better than the other. The lowest median value indicated the best performance.

V. COMPUTATIONAL EXPERIMENTS

The Tables in this section summarize the experiments results. A complete version of these data is available at http://www.dimap.ufrn.br/lae/en/projects/TSMPHOP.php.

The algorithms were implemented in C++ using the GCC compiler version 4.8.2 and executed on a server running CentOS 6.10 with 2 CPUs Intel Xeon E5-2670 @ 2.60GHz and 128GB of RAM. The solver executed on the same platform. The statistical tests were implemented in R using methods from [15]. The linearized mathematical model was solved by [8]. The execution used 32 parallel threads since the solver default configuration makes use of all available processors. For each problem, we gave 80,000s as a runtime limit for the optimizer to solve it.

The solver found the optimal solution of 32 instances of size 10 and 19 instances of size 20 from the *base* set. It did not find the optimal solution to any problem with 30 cities.

Table II shows the results of *unlimited* and *limited* experiments. The first column shows the name of the algorithm. The second and third columns show the results of the *unlimited* experiment. The other two columns relate to the *limited* experiments. The $\#(x_{min} \leq x_S)$ column shows the number of instances where x_{min} reached or bested x_S, a count related to the 96 instances from the *base* set. The Avg. T_m (s) column shows the average processing time (in seconds).

TABLE II
GENERAL RESULTS OF UNLIMITED AND LIMITED EXPERIMENTS

	Unlimited		Limited	
Algorithm	$\#(x_{min} \leq x_S)$	Avg. T_m(s)	$\#(x_{min} \leq x_S)$	Avg. T_m(s)
GA	22	93.1	15	30.7
MA	40	56.1	23	5.9
TGA	39	78.0	25	4.3
TGALS	42	85.0	25	4.6

Table II shows that the TGALS outperformed the other algorithms regarding the value of the best solution in the *unlimited* experiment. The MA presented the best processing time for the *unlimited* experiment. However, it found less best results than TGALS. The TGA and the TGALS obtained the best results on the *limited* experiment concerning solution quality and processing time.

The Friedman Aligned Ranks test pointed out significant differences for all groups. Tables III-VI summarizes the analysis by presenting the results of the Friedman *post-hoc* test with the Bergmann and Hommel's correction. Values exposed in those tables are the number of groups in which the *post-hoc* test pointed out significant differences (null hypothesis refutation). These values indicate the number of groups in which the algorithm in the line produced significantly better results than the algorithm shown in the column. The sixth column shows the sum of the values of each line. Since there are 24 instance groups and each pair of algorithms is compared, the maximum number of victories is 72, i.e., the value in the *Total* column can be, at most, 72.

Tables III and IV show the results concerning solution quality, i.e., the value of *G*, for the *unlimited* and *limited* experiments, respectively. These tables show that the TGALS exhibited significantly superior results in comparison to the other algorithms proposed with TGA being the second best.

Tables V and VI show the results concerning processing time for the *unlimited* and *limited* experiments, respectively. Tests regarding the *T* metric revealed that the MA spent less processing time followed by the GA in the *unlimited* experiments. In the *limited* experiment, the TGA performed better than the others, followed by the TGALS.

TABLE III
STATISTICAL ANALYSIS: *G* – UNLIMITED EXPERIMENT

	GA	MA	TGA	TGALS	Total
GA	-	0	0	0	0
MA	24	-	0	0	24
TGA	24	18	-	0	42
TGALS	24	21	14	-	59

TABLE IV
STATISTICAL ANALYSIS: *G* – LIMITED EXPERIMENT

	GA	MA	TGA	TGALS	Total
GA	-	0	0	0	0
MA	24	-	0	0	24
TGA	24	11	-	0	35
TGALS	24	13	2	-	39

TABLE V
STATISTICAL ANALYSIS: *T* – UNLIMITED EXPERIMENT

	GA	MA	TGA	TGALS	Total
GA	-	0	10	12	22
MA	20	-	24	24	68
TGA	10	0	-	6	16
TGALS	8	0	3	-	11

TABLE VI
STATISTICAL ANALYSIS: *T* – LIMITED EXPERIMENT

	GA	MA	TGA	TGALS	Total
GA	-	0	0	0	0
MA	24	-	1	0	25
TGA	24	22	-	2	48
TGALS	24	20	0	-	44

Results from the *unlimited* experiment indicated that algorithms based on the Transgenetic approach were capable of finding solutions with the better overall quality. Memetic and classic Genetic metaheuristics presented better performance considering processing times. In this *unlimited* scenario, the influence of the selected parameters' values tends to be higher since each heuristic has its particular stopping condition. By limiting the number of objective function evaluations, we set a common stop condition to reduce this effect. The *limited* experiment has demonstrated the prevalence of TGALS and TGA heuristics towards both solutions' quality and runtimes.

VI. CONCLUSION

As a new combinatorial optimization problem that directly addresses real-world aspects, TSMPHOP presents a set of attributes that place it in a prominent position regarding its applicability and solution complexity. The simple search for lower-cost cycles or routes where the vehicle is always at maximum occupancy are both ineffective in solving the problem, which requires carefully designed strategies to combine route and passenger loading aspects intelligently.

This paper presented the mathematical model for the TSMPHOP, which was entirely linearized and then implemented in a MIP optimizer [8]. Artificial instances were created to validate the performance of algorithms. Despite the solver being able to solver nonlinear constraints, we chose to work with the linear version of the model in hope to make it easier for the optimizer to find solutions within the time limit.

Four heuristics were developed: GA – a classic based on the vertical transfer of genetic material between parents and offspring; MA – that considers, besides the vertical transfer of genes, the interactions that occur between individuals of the same generation, also modifying the individuals' phenotype; TGA – an algorithm that mimics horizontal gene transfer between endosymbiont and host; and TGALS – an algorithm that adds phenotypic interactions to horizontal gene transfer.

To measure the performance of the developed algorithms, a computational experiment was carried out and its results were compared in light of statistical tools. TGALS was the algorithm that presented the best performance, demonstrating the effectiveness of the union between Memetic and Computational Transgenetic metaheuristic approaches for solving the TSMPHOP instances.

REFERENCES

[1] I. I. Melamed, S. I. Sergeev, and I. K. Sigal, "The traveling salesman problem," *Automation and Remote Control,* vol. 50, no. 9, pp. 1147-1173, 1989.

[2] M. R. Garey and D. S. Jonhson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. New York: WH Freeman, 1979.

[3] R. M. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, no. 1, pp. 45-68, 1975.

[4] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326-329, 1960.

[5] F. Glover and E. Woolsey, "Converting the 0-1 polynomial programming problem to a 0-1 linear program," *Operations Research*, vol. 22, no. 1, pp. 180-182, 1974.

[6] D. S. Chen, R.G. Batson, and Y. Dang, *Applied Integer Programming*. Hoboken, NJ: Wiley, 2010.

[7] D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms.* Redwood City, CA: Addison-Wesley, 1997.

[8] Gurobi Optimization LLC. *Gurobi Optimizer Reference Manual v7.5.2.* (2018). [Online]. Available: http://www.gurobi.com

[9] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498-516, 1973.

[10] Z. MICHALEWICZ, "Heuristics methods for evolutionary computation techniques," *Journal of Heuristics*, vol. 1, no. 2, pp. 177-206, 1996.

[11] E. F. G. Goldbarg and M. C. Goldbarg, "Transgenetic algorithm: a new endosymbiotic approach for evolutionary algorithms," *Foundations of Computational Intelligence*, vol. 3, pp. 425-460, 2009.

[12] M. S. Menezes, "Prize collecting traveling car renter problem: an algorithm study," Ph.D. dissertation, Universidade Federal do Rio Grande do Norte, Natal, Brazil, 2014. [Online]. Available: https://repositorio.ufrn.br/jspui/handle/123456789/18693

[13] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43-58, 2016.

[14] J. Derrac, S. García, D. Molina, and F. HERRERA, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no.1, pp. 3-18, 2011.

[15] B. CALVO and G. SANTAFÉ, "scmamp: statistical comparison of multiple algorithms in multiple problems," *The R Journal*, vol. 8/1, 2016.