RAIRO-Oper. Res. 58 (2024) 2239–2259
https://doi.org/10.1051/ro/2024084

RAIRO Operations Research
www.rairo-ro.org

TURKISH CASHIER PROBLEM WITH TIME WINDOWS AND ITS SOLUTION BY MATHEURISTIC ALGORITHMS

Ahmad Bassaleh* and Ekrem Duman

Abstract. Turkish Cashier Problem (TCP) is a new application area of the traveling salesman problem that was introduced to the literature recently. In this problem, the cashier can use public transportation or take a taxi where the cashier must visit multiple customer locations while minimizing the total transportation cost. In this study, we introduce a more realistic version of this problem where time has been integrated. This aspect is achieved by imposing time intervals within which the cashier must visit the customers. We name this problem as the TCP with time windows (TCPwTW). We develop several matheuristic algorithms to solve the TCPwTW: a modified version of the Simplify and Conquer (SAC) algorithm that was suggested for the TCP, simulated annealing (SA), original and modified versions of the migrating birds optimization (MBO) algorithm coupled with mathematical programming. We also tried to find the exact optimum using a Solver where for complex problems, only lower bounds were found. Numerical experimentation reveals that while for problems with loose time intervals, an exact solver can be considered. Once the time intervals tighten up, the best solutions can be obtained using matheuristics involving SA and MBO.

Mathematics Subject Classification. 90-05, 90B06.

Received October 7, 2023. Accepted April 4, 2024.

1. Introduction

The classical (standard, symmetric) TSP is a very well-known combinatorial optimization problem in literature and it is possible to see thousands of its applications [10]. TSP also has a lot of variants (a TSP-like problem formulated differently than the classical TSP) faced with in some real-life applications. One of these variants is the TSP with time windows (TSPwTW), where, in addition to the normal settings of the TSP, there is a time window defined by the earliest and latest arrival time associated with every customer. While minimizing the distance crossed by the traveling salesman, visiting the customers within their adequate time window becomes an additional obligation [11, 19].

One of the interesting applications of the TSP that has been introduced to the literature recently is the Turkish Cashier Problem [3] which is actually similar to the classical TSP with a special cost structure. In this problem, a cashier leaves his office in the morning and has to visit a number of customers to collect money. While traveling between locations, he can use public transportation (if there is any) or take a taxi. The cashier is asked to visit all locations with the minimum transportation cost possible. For the solution to this problem

Keywords. TSP applications, matheuristics, metaheuristics, migrating birds optimization, simulated annealing.

Industrial Engineering Department, Ozyegin University, Istanbul, Turkey.
*Corresponding author: ahmad.bassaleh@ozu.edu.tr

© The authors. Published by EDP Sciences, ROADEF, SMAI 2024

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

the author developed a simple solution algorithm named Simplify and Conquer (SAC) and obtained very good results on a number of test instances. In this study, we introduce a more realistic version of this problem where some customers are putting restrictions on when the cashier can visit them, and the cashier has to be back to the office latest at a specified time. We name this application as the TCP with time windows (TCPwTW).TCPwTW is actually a special case of the Vehicle Routing problem (VRP) with soft time windows and travel time and costs that was introduced by Hashimoto *et al.* [8]. In their setting, time windows can be violated, and travel times can be shortened with some extra costs. In this regard, our study becomes a real-life application of that study as well, where in their paper, they based their computational experimentation on the VRP with time windows (VRPTW) library.

Since even the classical TSP is an NP-Hard problem [13], for the large problem instances, we are often content with near-optimal solutions that are obtained by heuristic algorithms. In the broad sense, it is possible to categorize the heuristic algorithms for the TSP as constructive and improvement [7]. Constructive heuristics like nearest neighbor (NN), convex-hull (CH) [4] or SAC [3] build solutions step by step from scratch. On the other hand, improvement heuristics take a complete solution to the problem and try to improve it by small modifications.

Sometimes these modifications are applied in a simple manner, such as trying all possible modifications and terminating when no modification results in cost reduction, and sometimes, they can be applied in a more systematic way as part of metaheuristic algorithms. In metaheuristics the new solution obtained by a modification of the current solution is usually named a neighbor solution. It is possible to see the application of many different metaheuristic algorithms to the TSP and its variants [1,16,18,21,25].

We can mention the following works for SA implementations on TSP. Geng *et al.* [6] propose an effective local search algorithm based on simulated annealing and greedy search techniques to solve the TSP. In order to obtain more accurate solutions, the proposed algorithm, besides following the standard simulated annealing algorithm, adopts the combination of three kinds of mutations with different probabilities during its search. Then, a greedy search technique is used to speed up the convergence rate of the proposed algorithm. Rao [14] takes up the vehicle routing problem in a supply chain network, and after clustering, they solve the resulting TSP instances by SA and a genetic algorithm. Rao [20] formulates the distribution problem of an FMCG company as multiple TSP and solves it by the SA algorithm. Da Silva *et al.* [17] provides a thorough study of the performance of simulated annealing in the traveling salesman problem under correlated and long-tailed spatial scenarios. Duman and Duman[5] suggested going back to the best solution obtained so far in the last part of the iterations and allowing only downhill moves afterwards. This way they wanted to obtain a second chance to explore the valley containing the global optimum solution if ever the algorithm has skipped it before while accepting worse solutions.

As for the MBO for the TSP, Tongur and lker [23] developed and compared seven different neighborhood methods for the TSP and the ATSP. They showed that the performance of MBO can be increased by up to 36 percent with the right selection of the neighborhood method. In our study, we have implemented four different neighborhood search methods (vertex insertion, block insertion, block reverse, node swap) and have shown that although here, some methods are superior in use than others, a mixed use of them all enhances the performance of the algorithms.

Tonyal and Alkaya [24] applied MBO together with two other metaheuristics (SA and ABC (artificial bee colony algorithm)) on a special variant of the TSP. They implemented and compared 10 different neighborhood methods and found out that 2-opt performed the best for MBO. Duman and Duman [5] suggested two simple modifications on MBO to prevent any likely early convergence. The first modification was about not using the neighbor solutions borrowed from the solutions in front immediately but only after if there is a need. The second modification is about limiting the sharing of unused good neighbor solutions to only the immediate followers.

The matheuristics that have gained popularity recently utilize both mathematical programming and metaheuristic algorithms in their structure [22]. Since solving the original optimization problem optimally is too difficult, a solvable simplified version of it is sent to a solver. A heuristic or a metaheuristic algorithm fills in the gap between this problem and the original problem. In the literature, it is possible to find the use of

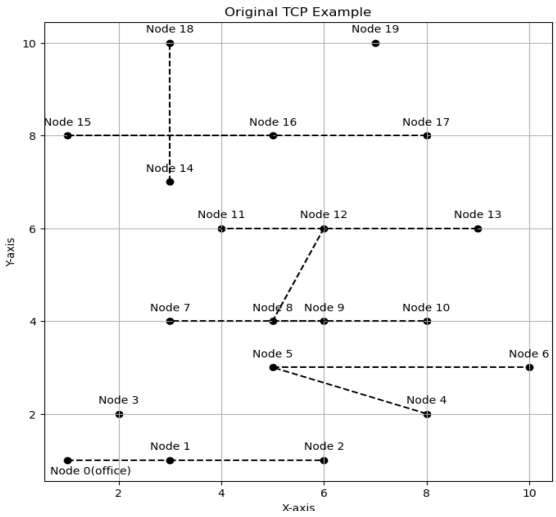

FIGURE 1. An example problem of the original TCP. The figure illustrates the distribution of customer locations in a 2d plane. Each dot represents a customer location and beside it is their index. Node 0 denotes the office. The dotted lines indicate the existence of public lines within certain node pairs.

matheuristics for various variants of the TSP. For example, Shahmanzari *et al.* [15] defined a new problem, which they name as the Roaming Salesman Problem, and solved it by a two-phase matheuristic approach. Lahyani *et al.* [12] addressed a rich Traveling Salesman Problem with Profits encountered in several real-life cases. They proposed a unified solution approach based on matheuristics and variable neighborhood search. He *et al.* [9] worked on flowshop group scheduling problem, which they approximated as a symmetric traveling salesman problem. Then, a matheuristic algorithm (MA) is proposed by integrating a branch-and-cut algorithm and an iterated greedy (IG) algorithm. Hashimoto *et al.* [8] used a kind of matheuristics as well. To find the vehicle routes, they deployed an iterated local search, and the service start times were decided optimally by dynamic programming implementation.

The contributions of this study are threefold. First, a new application of the TSP is introduced, and its mathematical formulation is given. Second, a modified version of the SAC algorithm, which was developed originally for the TCP, is developed. Third, a novel matheuristic approach is developed, coupling an exact solver and a number of metaheuristics algorithms, including the original and modified versions of SA and MBO.

The outline of the remainder of the paper is as follows. In section 2, we give a more detailed description and the formulation of the TCPwTW. Then, we explain our main solution approach based on matheuristics in section 3. The results of the numerical experimentation and the discussions on the results are provided in section 4. The paper is concluded in section 5 with a summary and indication of possible future research directions.

2. PROBLEM DEFINITION

In TCP, a cashier starts from his office in the morning and, after visiting a number of customers to collect money, returns to the office in the evening (see Fig. 1 where node 0 corresponds to the office of the cashier and nodes 1 to 19 are the customer locations). When traveling between location pairs, he can either take a taxi or use public transportation if the locations are connected with a public line (*e.g.* locations 1 and 2 in Fig. 1).

The objective of the cashier is to complete his operation with the minimum cost possible. TCP setting has been based on some assumptions which we assume are valid also for the TCPwTW:

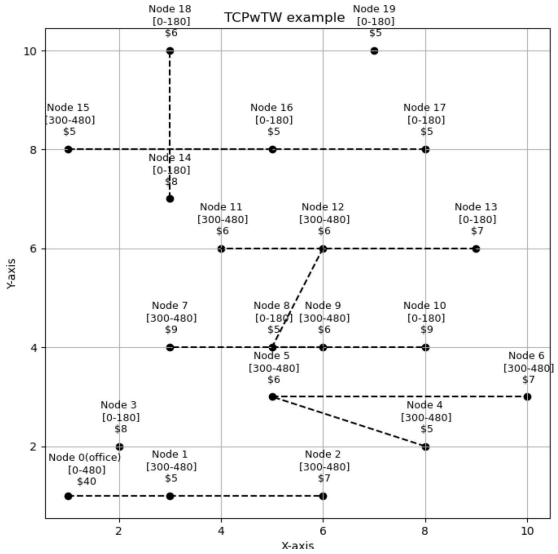

FIGURE 2. An example problem for TCPwTW. In addition to the illustrations shown in Figure 1, a time window (denoted in minutes) and a violation cost are associated with each node.

Visit sequence:	0	2	3	5	4	1	0
Travel Mode:	0	1	1	1	0	1	
Entry time (in minutes):	0	9.5	12	15	20	30	32.5

FIGURE 3. An example of a solution to the TCPwTW problem that includes its three components.

i) A previously visited node cannot be visited a second time even if it will be used as a cheaper connection to some other node.
ii) The cashier is allowed to skip some intermediate nodes and travel to a further point on the line. If he does so, he is not regarded as he visited the intermediate nodes.

In TCP, minimizing the cost of traveling using public transportation and taxis was the only aspect that was studied without taking time into account. To make the problem more applicable and extend it to problems faced in real life, here in this study, time aspect is also taken into consideration. Such an aspect is included by associating a soft time window (ranging in its tightness) that consists of the earliest time and latest time for every node in the network. The cashier must visit every node in the time window assigned to every node, neither earlier than the earliest time nor later than the latest (see the time values in square brackets near each location in Fig. 2). If a failure to do so occurs, a penalty must be paid.

Adding such constraint to the problem creates a trade-off between using taxis and public transportation. Public transportation is cheaper, while taxis are faster. Moreover, sometimes, even if a public line is offered when trying to reach a node from another, the utilization of a taxi might yield a cheaper overall solution since it will save the cashier time. With that time saved, no time window violation penalty costs will be applied to some nodes. Note that the cashier might take idle time throughout the trip. Such action can be taken as it can help in avoiding reaching certain nodes earlier than the earliest time associated with that node. As a result, the problem becomes finding i) The optimal sequence of visits between all the nodes, ii) The travel mode (public transportation or taxi) between every pair of nodes, and iii) The entry time of every node.

Figure 3 illustrates a solution example that is represented using three arrays. The first array starts and ends with node 0 (office). In the second array, 0 stands for public transportation whereas 1 stands for taxi. The third array shows the entry time (in minutes) of every node.

3. Solution framework

This section provides a detailed explanation of the methods used to tackle the TCPwTW, highlighting how each method is applied in solving the problem. We first introduce 1) an Exact solution, followed by 2) a Heuristic solution, and finally, 3) Metaheuristic solutions that integrate an exact solver into their iterative process (matheuristics). A discussion on the performances of these methods and suggestions on which one to use under which circumstances will be provided later.

3.1. Exact Solution: Commercial solver

As an exact approach, we present a mathematical model for the TCPwTW, which will be solved using an exact solver. With this method, we expect to find the optimal solution for small-scale problems and provide a lower bound (LB) for larger ones.

Mathematical model for the TCPwTW

An undirected graph $G = (V, A)$ is given, where $V = \{0, 1, 2, \ldots, n+1\}$ is the set of nodes and $A = \{(i, j) \mid i \neq j; i, j \in V\}$ is the set of undirected arcs. Node 0 and $n+1$ refers to the office, while the remaining nodes corresponds to locations that must be visited. Decision variables x_{ij} and y_{ij} for $i, j = 0, 1, \ldots, n$ decide on the order of location visits the cashier must follow. where x_{ij} or y_{ij} indicate if the cashier travels from location i to location j using a taxi or public bus respectively by adapting the value of 1, and 0 otherwise. A travel cost associated with traveling from location i to j using a taxi denoted as c_{ij}, and a time duration for the trip represented with t_{ij} for $i, j = 0, 1, \ldots, n$. Likewise, a cost is associated with using public buses exist and is expressed with m_{ij}, while p_{ij} shows the corresponding traveling time. Each location has a soft time window constructed using an early e_i and a late l_i time for $i = 0, 1, \ldots, n+1$. f_i represents the penalty of visiting node i outside its time window. Decision variables t_i for $i = 0, 1, \ldots, n+1$ indicates the visiting time of each location and decision variables k_i for $i = 0, 1, \ldots, n+1$ examines whether location i's time window was violated by taking the value of 1 if i's time window has been violated, 0 otherwise. M is a sufficiently big number that aids in the formulation of certain constraints.

Objective Function

$$\textbf{min} \quad \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij} + \sum_{i=0}^{n}\sum_{j=0}^{n} m_{ij}y_{ij} + \sum_{i=0}^{n+1} f_i k_i \tag{1}$$

Constraints

$$\sum_{i=0}^{n} x_{ij} + \sum_{i=0}^{n} y_{ij} = 1 \qquad j = 0, 1, \ldots, n \tag{2}$$

$$\sum_{j=0}^{n} x_{ij} + \sum_{j=0}^{n} y_{ij} = 1 \qquad i = 0, 1, \ldots, n \tag{3}$$

$$t_j - t_i - t_{ij}x_{ij} - p_{ij}y_{ij} \geq M(x_{ij} + y_{ij} - 1) \qquad i = 0, 1, \ldots, n; j = 1, \ldots, n \tag{4}$$

$$t_{n+1} - t_i - t_{i0}x_{i0} - p_{i0}y_{i0} \geq M(x_{i0} + y_{i0} - 1) \qquad i = 1, \ldots, n \tag{5}$$

$$t_0 = 0 \tag{6}$$

$$t_i \geq e_i - Mk_i \qquad i = 0, 1, \ldots, n+1 \tag{7}$$

$$t_i \leq l_i + Mk_i \qquad i = 0, 1, \ldots, n+1 \tag{8}$$

$$t_i \geq 0 \qquad i = 0, 1, \ldots, n+1 \tag{9}$$

$$k_i \in \{0, 1\} \qquad i = 0, 1, \ldots, n+1 \tag{10}$$

$$x_{ij} \in \{0, 1\} \qquad i = 0, 1, \ldots, n; j = 0, 1, \ldots, n \tag{11}$$

$$y_{ij} \in \{0, 1\} \qquad i = 0, 1, \ldots, n; j = 0, 1, \ldots, n \tag{12}$$

The objective function (1) aims to minimize the total cost resulting from using taxis and public buses to visit all the locations and return to the office (depot), in addition to minimizing the total penalty costs occurring. Through (2) and (3), it is assured that every location is entered to, and left from exactly once. (4) ensures that if node j is visited after node i, the time node j is visited is greater than or equal to the time node i was entered plus the time it took to reach node j from node i. (5) ensures the same but for the returning time of the office depot. (4) and (5) also ensures that no sub-tours are formed while constructing a route. Through (6), we ensure that the cashier starts his trip from the office. (7) ensures that every node is visited at a time not earlier than its open (early) time and if preferred to do so, a cost must be paid. (8) obligates the same but for the nodes' closing (late) time. Finally, (9) imposes the continuous behavior of decision variables t_i, and (10), (11), and (12) imposes the binary condition to the rest of the decision variables.

3.2. Heuristic solution: modified SAC (mSAC)

The Simplify and Conquer (SAC) algorithm is a heuristic approach that leverages the special structure of the Turkish cashier problem (TCP) to its advantage. The algorithm begins by grouping customer locations into subsets that possess interconnected public line connections. Any pair of locations within each subset can be reached using a public connection line. This type of clustering is implemented in hopes of maximizing the number of times a bus connection is used over taxis.

Every subset is then treated as a node that must be visited, converging the problem as a whole to a TSP. The distance between every pair of subsets is assumed to be the distance between the closest customer locations from each of the two subsets.

The cashier then proceeds to visit the subsets while obeying the sequence provided. Within each subset, the grouped locations must all be visited given a predefined set of rules that aim utilizing the public lines as much as possible. Once the cashier is done with visiting all the locations inside a subset, he moves to another unvisited subset using a taxi. More about the implementation of SAC can be found in [3].

The original version of the SAC algorithm prioritizes cost efficiency in route construction without taking the travel time into consideration. Such a trait significantly hinders the algorithm's ability to achieve an optimal or suboptimal solution, as the time window penalties (both early and late) hold great importance in determining every route's cost. In mSAC, as an effort to incorporate time constraints into the algorithm, a swap process is implemented. The swap moves entails evaluating the cost that results after exchanging the visiting order of a node with an early time window violation with a node that has a late time window violation. The selection of the swap move is based on identifying the exchange that yields the greatest reduction in cost. This process iterates until no further swap move can be found that results in a reduction in the overall cost of the solution.

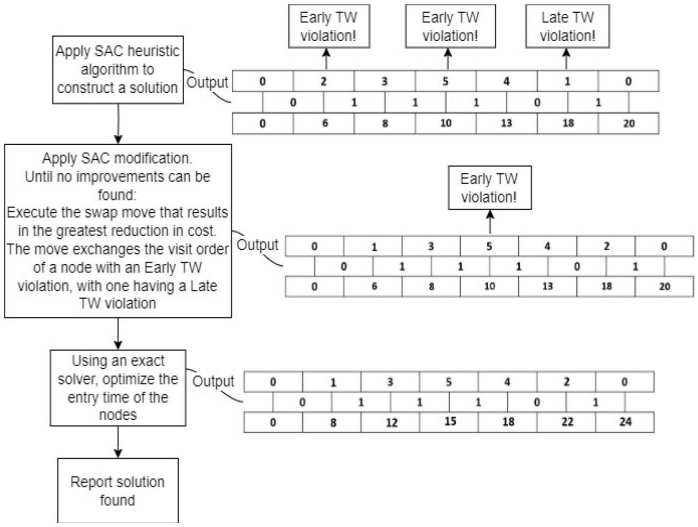

FIGURE 4. mSAC implementation scheme. The output represents a sample of how a solution looks like after each step is applied.

The guiding principle behind this modification is that by exchanging a node that has an early violation with one that has a late violation, we introduce the potential to address time discrepancies more effectively, enabling the arrival at nodes with early violations within later time frames, and vice versa. Note that the solutions produced are always feasible. Each customer is visited exactly once using either a taxi or a public bus, and the timing of each customer visit is kept logical. To reach a node from a previous one, we add the time spent at the first node to the time required to reach the next node.

The modified SAC (mSAC), together with the original SAC can produce the first two arrays of a complete TCPwTW solution. To obtain the third array we may assume that the time each node must be entered can be calculated as follows: when node j is the next destination, the time it will be entered equals the current time plus the travel time from the current node to node j. This assumes the instantaneous behaviour of the cashier while traversing node pairs. However sometimes, incorporating some idle time into the trips can generate cheaper solutions since this can help in reaching certain nodes later than their early time window restriction.

To address this matter, the optimum entry time to each location is optimally decided on using a simplified version of the TCPwTW mathematical model, where the basic difference is that the visiting sequence and travel mode information are fed as parameters to the model after running the mSAC algorithm. Here, the decision variables x_{ij} and y_{ij} of the model are treated as parameters. Given the visiting sequence and travel modes, the optimal entry time of the nodes are decided on almost instantaneously using the exact solver. See Figure 4 for a schematic presentation of the mSAC algorithm.

3.3. Matheuristics involving the SA and the MBO algorithms

Metaheuristic algorithms leverage the use of neighborhood search strategies to systematically explore different areas in a solution space. Usually, a new solution is found sufficiently by fully depending on the neighborhood search strategies within a metaheuristic. Although, we adapted another approach in this study. We first go over how the neighborhood search step is operated within the metaheuristics used, then we explain each algorithm in detail.

Metaheuristic algorithms integrated within a matheuristic framework can be designed to produce either only the first (visiting sequence) or the first two arrays (visiting sequence and travel modes) of a TCPwTW solution. It is possible to see both kinds of designs in the literature. Attempting to obtain all three arrays from

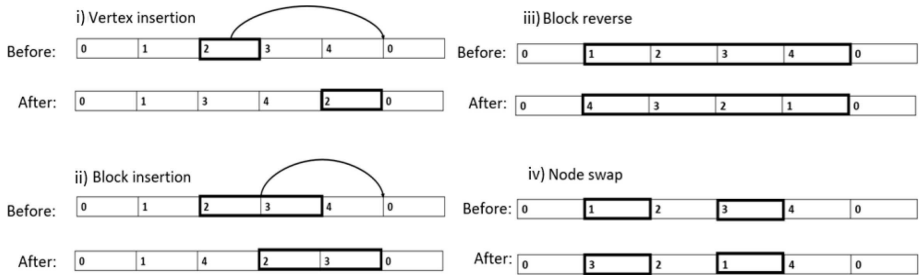

FIGURE 5. The Neighborhood Search strategies.

a neighborhood search would be too complicated if not impossible. In this regard, we preferred to obtain only the first array using the neighborhood search strategies (will be discussed next) of the metaheuristic algorithms, and let an exact solver decide on the optimal values of the second and third arrays given the values of the first array fixed.

The changes to the TCPwTW model will be that the visiting sequence obtained by the metaheuristics is provided as parameters to the model. This becomes possible by adding a binary parameter denoted as z_{ij} for $j = 0, 1, \ldots, n$, where z_{ij} takes the value of 1 if the sequence fed from a metaheuristic indicates that location j must be visited after location i, 0 otherwise. The model is forced to follow the sequence provided by including the following constraint to it:

$$x_{ij} + y_{ij} = z_{ij} \qquad i = 0, 1, \ldots, n; j = 0, 1, \ldots, n$$

For the node pairs where $z_{ij} = 1$, the solver will then decide which of the two transportation alternatives should be used: taking a taxi ($x_{ij} = 1$) or using a public bus ($y_{ij} = 1$). The essence of this approach is to optimally solve sub-problems, with the hope of obtaining better solutions to the overall problem.

In the following sections, we describe the Simulated Annealing (SA) and the Migrating Birds Optimization (MBO) algorithms. Prior to delving into them, we describe the four distinct neighborhood definitions and search strategies that we have incorporated and employed within the metaheuristics.

3.3.1. Neighborhood Search strategies implemented

Numerous neighborhood search strategies have been developed to address optimization challenges featuring permutation-type solutions, notably exemplified by the TSP [23]. The performance of these strategies can vary depending on the structure of the problem being targeted. In this study, we focused on four different strategies, and examined their performance on the problem at hand. These strategies are: i) Vertex Insertion, ii) Block Insertion, iii) Block Reverse, and iv) Node Swap. Below is a description of each of the methods mentioned (a graphical explanation is given in Fig. 5):

i Vertex insertion (VI): A point is selected randomly. This point is removed from its position in the array provided and relocated to another random position.
ii Block Insertion (BI): A subset of a random size is selected from an array provided. This subset is then removed from its original position and relocated to another position chosen randomly within the array.
iii Block Reverse (BR): A subset of a random size is selected from an array provided. While keeping the subset in its same position, the order the points follow within the subset is reversed.
iv Node swap (NS): Two points are selected randomly within an array and the locations of them are swapped.

A crucial question to ask here is which of the strategies must be adapted in the neighborhood search stage of the metaheuristics. While a straightforward approach would be adapting a single strategy throughout the algorithms, an alternative approach entails the inclusion of a variety of different strategies. In this study, the

TABLE 1. Results of different neighborhood search strategy combinations.

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**16**
VI	100%	0%	0%	0%	50%	50%	50%	0%	0%	0%	33%	33%	33%	0%	25%	**45%**
BI	0%	100%	0%	0%	50%	0%	0%	50%	50%	0%	0%	33%	33%	33%	25%	**10%**
NS	0%	0%	100%	0%	0%	50%	0%	50%	0%	50%	33%	0%	33%	33%	25%	**5%**
BR	0%	0%	0%	100%	0%	0%	50%	0%	50%	50%	33%	33%	0%	33%	25%	**40%**
Result	8.0%	11.2%	11.1%	7.2%	4.4%	7.3%	2.1%	10.7%	5.6%	6.6%	3.3%	4.8%	5.5%	5.2%	1.9%	**0.5%**

latter was chosen based on two motivations. The first one is that it entails creating a more diverse search mechanism. Diversification is a crucial element that must be maintained while searching in the neighborhoods of the solutions [23]. At some points throughout an algorithm, the full neighborhood of a solution can be explored, leaving it stranded at a local optimum. Infesting the algorithm with unique strategies facilitates reducing the frequency of facing such phenomena. The second reason behind our approach is inspired from [6] where they faced a similar study. In context of neighborhood search strategies, methods i, ii, and iii were also utilized. They questioned how well each strategy performs when it is used alone, and with the other ones. Subsequently, they tested the algorithm on the same problem using different combinations of the neighborhood strategies. This left them off with a table that explains the CPU time and the solution quality each combination yields. Their findings underscored the superiority of employing a blend of all strategies, while assigning higher probability of selection to the strategies that yielded the highest returns, as the selection of them is preferred throughout the search.

Consequently, we conducted an experiment to assess the performance of the four strategies on our problem, as summarized in Table 1. Similar to [6], we examined the performance of all possible combinations out of the four methods by measuring their percentage distance from the optimal solution, given the same problem instance. The results helped us answer two questions. The first one is which of the methods should be utilized. As seen in Table 1, using a combination of all the methods yielded the best results (test 15), suggesting that the employment of all four methods is preferred throughout the search. The second question is how often each method must be chosen. By observing the performance of the four methods when used individually (tests 1-4), it is evident that VI and BR achieved the best results, while BI and NS fell behind. This indicates that the use of VI and BR throughout the search can be preferred than BI and NS. This again can be asserted by looking at the results of the combination of VI and BR (test 7) and the combination of BI and NS (test 8). As a result, we decided to distribute probabilities having a factor of 5 (for simplicity), with the lowest likelihood assigned a value of 5% and for the Node swap method, followed by the second least favorable method at 10% claimed by Block insertion method. Conversely, the highest probability was attributed 45% for Vertex insertion, with the second-best method closely behind at 40% and for Block reverse method. In comparison with using the four methods having equal frequencies (test 15), the adapted probability assignment values enabled us to advance 1.4% closer to the optimal solution (test 16), establishing it as the most effective frequency assignment values discovered.

The whole neighborhood search scheme mentioned above is applied every time the neighborhood search method within the metaheuristics is called to generate a new solution. Figure 6 is presented to demonstrate the mechanism of the scheme.

Similar to mSAC, when providing the exact solver, a model with the most crucial decisions as parameters, the problem is optimally solved in an insignificant amount of time (milliseconds).

3.3.2. Simulated annealing (SA)

The algorithm starts with an initial solution to the problem. It then gradually explores the solution space by generating neighboring solutions and deciding whether they should be accepted or rejected. The algorithm's

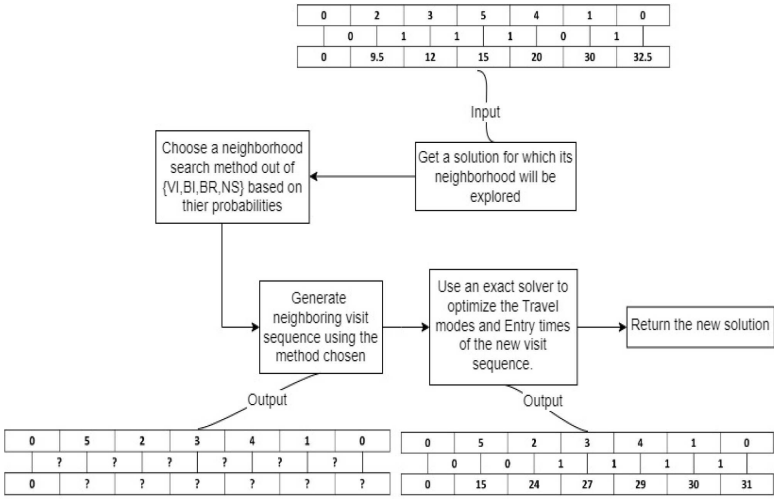

FIGURE 6. Operations conducted within a neighborhood search method. The input & output represents a sample of how a solution looks like after each step is applied. In this example, NS method is chosen, where the visit order of node 2 and 5 is swapped.

behavior is influenced by a temperature parameter that controls the search behavior, favoring explorations at the early stages of the algorithm while favoring exploitation at the final stages. Below is a step-by-step implementation of SA.

1. Initialization (set the algorithm's parameters):
   - T_i (initial temperature): The temperature the algorithm starts with
   - T_f (final temperature): The temperature during the last iterations
   - L (epoch length): Indicates how many iterations will be performed at every temperature level
   - α (cooling rate): A factor that takes a value between 0 and 1, used to reduce the temperature
   - S_c (current solution): Initialize the current solution using a heuristic algorithm or randomly.
   - S_b (best solution so far): Initialize the best solution as S_c.
2. Main loop:
   We generate a new solution neighboring S_c using the neighboorhood search scheme mentioned earlier. The neighbor will be adapted as S_c if it shows a better objective value (lower cost in our case). If it is also better than the best solution found, we also assign it as S_b. If it has an objective value that is worse than S_c, we adapt it randomly based on the Metropolis criterion where the acceptance probability is larger at high temperatures and at low differences with the current objective value. The neighbor solution is discarded if it fails the criterion, while keeping S_c unchanged. The Metropolis criterion is defined as follows:

$$S_c = \begin{cases} neighbor & \text{if } \text{rand}(0,1) \leq \exp\left(-\frac{E(\text{neighbor})-E(S_c)}{T_i}\right) \\ S_c & \text{otherwise} \end{cases}$$

   After this operation is completed L many times, T_i is scaled down by multiplying it by α. The algorithm keeps iterating until T_i reaches a value smaller than or equal to T_f. Below is a pseudocode illustration of SA's main loop:

Algorithm 1: SA Main Loop Pseudocode.

```
while T_i ≥ T_f do
    for L iterations do
        Generate a new neighboring solution of S_c;
        if new solution is better than current solution then
            S_c = new solution;
            if S_c is better than S_b then
                S_b = S_c;
        else if new solution passes the Metropolis criterion then
            S_c = new solution;
    Update the temperature: T_i = T_i * α;
```

3. Report best solution (S_b) found

SA Parameters

To determine the initial temperature (T_i), we used the same methodology of Duman and Duman [5]. Their approach gives a control mechanism that avoids entering terrible search spaces at the beginning stages of the algorithm. The algorithm can significantly struggle if it starts off at a valley far apart from the optimal solution. To address this, a reference solution that has a fairly good cost is established, from which we measure a certain percentage deviation. In this context, the solution obtained by mSAC can serve as a suitable reference point. To be more specific, mSAC is run first, and the cost of the solution found is saved. Afterwards, we set the initial temperature for SA in a way such that solutions with costs deviating by 10% from the mSACs solution cost have about 50% chance of getting accepted at the early stages of the algorithm.

Regarding the epoch length (L) operated at each temperature level, there was no specific value imposed. Instead, the algorithm operates at each temperature level until reaching a predefined time limit. This time limit is determined by dividing the total given run time of the algorithm by the number of iterations it will take for the initial temperature (T_i) to reach the final temperature (T_f). We can find how many times T_i must be scaled down by the alpha (α) factor to reach T_f using the following equation:

$$\text{Number of iterations} = \frac{\ln(T_f/T_i)}{\ln(\alpha)}$$

This method is preferred to be followed to set a limit for L since we are comparing our methodologies based on a stopping time criteria. Moreover, the approximation of the parameters of SA that will allow it to terminate at that exact predefined time is not possible. By limiting how long each temperature level can operate for, we assure the termination of the algorithm at the desired time limit.

Regarding an initial solution to set S_c with, a randomly generated solution was used. To find the rest of SAs parameters, partial enumeration experimentation was used. These experiments involve systematically incrementing certain parameters while fixing others, thereby enabling a thorough assessment of the impact of parameter variations on both the objective function and the algorithms runtime. The outcomes of our experimental investigations indicate an optimal value for α of 0.975, allowing a balance between the exploration and exploitation phases. As for the T_f setting, a value of 0.005 was used.

Modification on SA

Recently, a modification to SA was suggested by Duman and Duman [5] which includes shifting entirely from the exploration of new neighborhoods to exploiting the best solution found when x% (say 90%) of the algorithm is completed. It is commonly assumed that after a significant number of iterations, the solution resides in one of the favorable valleys in the search space, potentially encompassing the global optimum. However, there is a possibility that despite being on the path leading towards the global optimum, transitioning to a different valley due to the acceptance of inferior solutions can occur. Thus, returning to the best solution discovered thus

far after a certain percentage of total iterations has been completed and dedicating the remaining iterations exclusively to improving moves makes sense.

3.3.3. Migrating Birds Optimization (MBO) algorithm

The MBO algorithm is inspired by the communal behavior (V-shape) of bird flocks during migration. The algorithm is based on the observation that migrating birds exhibit efficient navigation and communication strategies to reach their destinations. Below, we demonstrate the general flow of the MBO algorithm. Note that within the MBO terminology, every solution corresponds to a bird, and conversely, each bird corresponds to a solution. Therefore, we will interchangeably use these two terms.

1. Initialization (set the algorithm's parameters and construct a V flight shape):
   - n: The number of initial solutions
   - k: The number of neighboring solutions to generate for the leader
   - x: The number of neighbors to pass downwards
   - m: The number of iterations maintaining the same leader solution
   - K: The iteration limit where each neighbor solution generated counts as an iteration.
   - V flight formation: arrange n many initial birds in a V-shaped structure, with one bird positioned at the tip, and others of equal size positioned symmetrically on both the left and right sides.
2. Main loop:
   During each tour, we discover k new solutions neighboring the leader bird using the neighborhood scheme detailed earlier in this section. If a new solution surpasses the leader bird in performance (in our case, achieving a lower cost), we replace the leader bird with the most superior neighbor. Then, we create two distinct lists, each containing x unused neighbors derived from the leader bird. One list is designated to be passed to each of the right and left sides. We then proceed to individually iterate over each bird on both sides.
   For each bird, we seek out k-x new neighboring solutions. Using the neighbor solutions and the list of passed solutions, we try to improve the current bird. Afterwards, we pass the best unused neighbors from the combined list down to the birds that follow.
   After this process is completed down to the last bird of both sides, a tour is completed. After repeating this operation m times, a leader change operation occurs. The leader bird relocates to the lowest position on one side, with the first bird from that side assuming leadership.
   The algorithm terminates after this operation is completed K many times. Below is a pseudocode of MBO algorithm's main loop:

Algorithm 2: MBO Main Loop Pseudocode.

```
for K iterations do
    for m iterations do
        Generate k neighbors using the leader bird.;
        if best neighbor outperforms leader bird then
            Set the best neighbor as the leader bird;
        end
        Initialize empty lists: leftPass, rightPass;
        Save x neighbors from the leader bird into leftPass and rightPass;
        for each bird on the left side do
            Generate k-x neighbors using the current bird.;
            Find the best solution from {neighbors, leftPass};
            if best solution is better than current bird then
                Update the current bird to the best solution;
                Update leftPass;
            end
        end
        for each bird on the right side do
            Generate k–x neighbors using the current bird.;
            Find the best solution from {neighbors, rightPass};
            if best solution is better than current bird then
                Update the current bird to the best solution;
                Update rightPass;
            end
        end
    end
    Move the leader bird to one side, with the first bird from the same side becoming the new leader;
end
```

3. Report best bird found

MBO algorithm Parameters

To gain a nuanced understanding of the parameter values that contribute to the enhanced performance of the MBO, we employed a methodology similar to the approach used for some parameters within the SA algorithm. Here again, a series of partial enumeration experiments were conducted. Through careful analysis of the insights gained from the partial enumeration experiments, it was possible to approximate and recognize the parameter values that hold the greatest potential for the algorithms utilization.

We concluded the use of 11 birds as the number of initial birds (n). These initial birds are generated randomly. The number of neighbors to generate (k) is estimated as 5, and the number of birds to pass (x) is determined as 1. As for the number of tours (m), the use of 1 tour proved to be the best. The number of iterations to perform (K) is not fixed as our algorithms are set to run until a predefined time limit is reached.

Modifications on MBO

The original MBO might suffer from early convergence. Recently Duman and Duman [5] has suggested two modifications to decrease the likelihood of this outcome.

Modification 1: One-step sharing

The first modification involves the list of solutions that can be passed down to the succeeding birds, where only the most recent solutions discovered can shared. Different from the conventional approach of drawing from the cumulative pool of preceding solutions, this approach introduces a noteworthy behavior that has the potential of enhancing the diversity of exploration within the solution space. By embracing One-step sharing, the birds enhance their ability to navigate the complex solution space. The limitations of the traditional approach might restrict the exploration to a more limited subset of potential solutions.

Modification 2: Delayed evaluation

An additional noteworthy modification involves granting the birds the privilege to adopt solutions generated by their preceding birds only if none of their neighboring solutions surpasses the quality of their own existing solution. This particular enhancement again serves as a strategic measure aimed at broadening the exploration within the extensive search space. By embracing the Delayed evaluation modification, the flock demonstrates a heightened ability to balance between local exploitation and global exploration, effectively boosting the overall search capabilities and optimizing the discovery of promising solutions across the search space.

4. Numerical analysis

In this section, we first explain the generation logic of the data we used in this study, which is followed by the results obtained and discussions on them.

4.1. Experimentation Data

The necessary parameters for the problem discussed in this paper are as follows:

i. The x and y coordinates of each node in the network
ii. The cost of traveling using a taxi or public transportation between any two nodes in the network
iii. The time it takes to travel between every two nodes in the network using a taxi or public transportation.
iv. The early and late time window of each node in the network
v. The penalty that must be paid for every node if its time window is violated.

We now provide an explanation of how each of these particular data was generated. Regarding the x and y coordinates of each node in the network, the utilization of some instances with different node sizes was taken from [3].

Afterwards, the Euclidean distance is calculated between every two points in the network. Once the Euclidean distance was found, it was possible to set a traveling cost that applies whenever traversing a node pair using taxis and public transportation. The cost of traveling using a taxi is calculated as a fixed cost of 1 unit plus the distance between nodes multiplied by the unit cost of traveling (assumed to be 1 unit). The 1 unit fixed cost in taxis is included as a way to reflect a start-up cost. Regarding the use of public transportation, if a public line exists between a pair of nodes, then the cost of traveling using that line is taken as one unit. Otherwise, it is assumed that there is a huge (big M) cost occurring when attempting to travel between these pair of nodes *via* public transportation.

Next, we assigned the time it takes to travel between every pair of nodes equal to the distance between the pair divided by the speed of the travel mode being taken. The speed of the taxi is assumed to be twice that of public transportation. Such speed settings were maintained to demonstrate the case that can be faced in real life. Public transportation must make numerous stops while traveling to pick up and drop riders, which makes reaching nodes more time-consuming. If a public line does not exist between any node pair, then attempting to use that travel mode will take a big M amount of time (similar to the traveling cost).

Regarding the generation of early and late time windows for the nodes in any given network, three different settings with different tightness levels were generated, which we name as easy, medium, and hard. The easy time window only obligates the visit of each node before the working hours of the day are over, leaving a big chance for the solutions to meet all the nodes' time windows. Regarding the medium time windows, two time windows were generated and randomly assigned to the nodes in the network. Those time windows had three hours of tightness, and the two time windows came as i) between 9 am and 12 pm (morning shift), ii) between 2 pm and 5 pm (afternoon shift). Finally, for the hard time windows, four time windows were generated and again randomly assigned to the nodes in the network. Those time windows had two hours of tightness and came as i) between 9 am and 11 am, ii) 10 am and 12 pm, iii) 2 pm and 4 pm, and iv) 3 pm and 5 pm. We did not integrate an available time window between 12pm and 2pm as we assumed that the customers were on a lunch break during these times.

Three types of time windows for each instance were generated to study the complexity between loose and tight time windows, and to see how the exact solver handles these problems.

In the context of establishing time window violation penalties for each node, it is crucial to ensure their competitiveness with travel costs. If the cost is set too low, the impact of time complexity on determining optimal solutions will be negligible, as the focus will primarily be on identifying the cheapest routes. Likewise, if the cost is set too high, time complexity will overshadow the traveling cost considerations, and successful solutions will prioritize satisfying all node time windows rather than finding cheap routes. After having an overview of the cost produced without having time windows on the nodes, assessments were made to keep the time window violation cost almost proportional to the travel costs. It was determined to use a random cost between 5 and 9 for each node that had its time window violated. The noted settings are applicable to all nodes except for the office. In practice, the cashier adheres to specific working hours, typically limited to 8 hours. In order to incorporate this working scheme into our problem, the office operates within a time window from 9:00 am to 5:00 pm, and a penalty cost of 40 is imposed if the cashier arrives late to the office. This increased penalty has been established to account for the inclusion of the designated working hours of the cashier, where any overtime spent by the cashier implies that they have worked beyond their designated hours which is not permittable in real life.

This paper aims to target problems that pose equal challenges in maintaining cheap routes while simultaneously meeting nodes' time windows, as these scenarios present the most intricate problem-solving complexities.

4.2. Results and discussions

Seven instances of size 20 varying in node locations, available public connection lines, time window violation cost, and 3 types of time windows ranging from easy to hard tightness were utilized to test the methodologies discussed.

Three instances of size 50 with the same node locations and available public connection lines, but varying in time window violation cost, one easy TW (Time window) and three different medium and hard TWs are also used to generate multiple more instances and dive deeper into understanding the complex factors of the problem.

The methods that were experimented on the set of problems are mSAC, SA with the modifications mentioned, the original MBO, MBO with the delayed evaluation, MBO with one-step evaluation, and finally MBO with both delayed evaluation and one-step sharing modifications. All these methods' output is compared against an exact solver's solutions. The solver can report the optimal solution of a given problem if it is found within the given runtime limit, or report the best solution found up to the run time limit, along with a lower bound that may or may not be reached.

To evaluate the performances of the matheuristics against the exact solver, for each problem, the matheuristics and the solver are given the same run time to keep the comparison fair. As the problem size increases and its nodes' time window gets tighter, the dedicated run time is increased assuming that the complexity of solving them also increases thus, more run time is needed. As for mSAC, its outcomes are provided instantaneously since it is a heuristic approach with several deterministic rules.

The matheuristics used yield different solutions every time they are run due to the randomness involved. To reduce this variability, five runs were made on each algorithm for each instance, and the average of those five runs is recorded for comparison.

All methods described in this study were implemented using Java programming on a Monster Abra A5 V19.2.5 laptop, with Core i5-12500H 2.50 GHz and 16GB RAM. To solve problems optimally, Gurobi optimization software that had version 10.0.1 was used.

First, we provide the general results obtained by each method on each instance in Table 2. Afterwards in Table 3, we compare the performance of the heuristic and metaheuristic algorithms against the solver's performance by noting how much the results found deviate from the ones obtained by the exact solver (percentage wise) given a time limitation. Following that in Table 4, we compare all the methods with the lower bound found

TABLE 2. General results.

Problem	n	TW	Time	LB	Solver	mSAC	SA	MBO	Del	1Step	Del+1Step
A	20	easy	45	34.20	34.20	85.30	34.40	34.80	34.60	35.50	34.40
B	20	easy	45	38.90	38.90	115.50	43.40	39.20	39.70	39.00	39.20
C	20	easy	45	41.70	41.70	42.90	41.70	41.80	41.90	42.00	41.80
D	20	easy	45	44.40	44.40	45.20	44.50	44.50	45.20	44.70	44.60
E	20	easy	45	44.10	44.10	111.50	45.60	45.30	46.00	45.80	45.30
F	20	easy	45	32.00	32.00	131.30	35.80	33.50	33.40	33.20	32.60
G	20	easy	45	31.20	31.20	88.80	31.70	31.80	31.30	31.60	31.30
H1	50	easy	150	67.60	72.20	232.90	108.70	94.80	93.70	98.40	97.10
A	20	medium	60	68.80	80.10	148.10	82.90	82.70	83.30	82.10	84.80
B	20	medium	60	59.60	59.60	137.90	59.80	60.70	61.00	59.90	59.60
C	20	medium	60	56.40	62.80	86.30	63.60	67.20	64.50	64.90	64.00
D	20	medium	60	62.40	72.30	98.00	70.70	70.80	68.90	70.20	72.10
E	20	medium	60	63.00	69.90	133.20	71.20	71.70	73.90	72.10	71.40
F	20	medium	60	49.70	73.70	147.30	67.10	68.60	70.10	68.40	68.20
G	20	medium	60	48.60	57.50	114.40	59.00	57.60	57.70	57.50	57.70
H1	50	medium	700	90.90	202.90	362.60	153.30	173.70	169.40	169.50	178.20
H2	50	medium	700	92.90	207.10	359.70	156.30	182.70	176.40	175.30	179.90
H3	50	medium	700	92.50	195.80	364.80	160.70	168.40	157.20	157.30	175.70
A	20	hard	90	62.70	111.90	143.70	89.70	86.60	86.90	87.00	88.10
B	20	hard	90	58.40	98.70	173.50	100.60	99.30	98.20	99.40	98.60
C	20	hard	90	65.40	83.30	100.90	86.10	84.10	84.20	84.20	86.20
D	20	hard	90	60.50	76.10	101.80	78.20	76.60	78.30	77.30	77.50
E	20	hard	90	63.20	80.70	139.50	80.90	81.40	81.40	80.70	81.40
F	20	hard	90	57.90	92.50	183.40	88.80	86.60	89.30	87.50	88.80
G	20	hard	90	49.10	64.70	122.00	70.00	68.40	70.40	69.90	69.20
H1	50	hard	1200	91.50	254.30	388.40	183.70	216.50	195.00	204.50	195.30
H2	50	hard	1200	91.50	319.50	414.90	213.80	219.50	213.10	218.70	211.40
H3	50	hard	1200	92.80	273.50	378.80	188.10	212.20	191.90	198.80	198.70

Notes. *Notations*
Problem: tested problem title, n: number of nodes, TW: the difficulty of the time window, Time: run time limit (in seconds), LB: Lower bound obtained by the solver, Solver: Solver results, mSAC: modified SAC results, SA: SA results, MBO: The original MBO algorithm results, Del: MBO with delayed evaluation results, 1Step: MBO with one-step sharing results, Del+1Step: MBO with one-step sharing and delayed evaluation results.

by the solver for each instance. Finally, we present a dominance matrix that identifies which method dominated which other in Table 5. All the data generated and used in this paper can be found in [2].

The mSAC failed to adequately yield high-quality solutions as its results suggest that it still fell behind in integrating the time factor. SA and MBO with its variants, on the other hand proved their use as they frequently provided solutions close to if not better than the ones obtained by an exact solver given a time limitation.

Exploring the different versions of MBO, for problems with the smallest number of nodes, no conclusive evidence supports one variant over another, as their outcomes were closely aligned. The variants did not have enough time to demonstrate their diversity in exploring the search space which resulted in similar behavior.

When shifting to problems with larger node sizes, observing the uniqueness of the variants becomes clearer as more running time becomes permitted. The original version of the MBO performed better than some variants of it with the problem given the least running time (soft time window problems) as it was able to explore regions in the search space faster, allowing it to stumble upon solutions the fastest. However, once the time limit becomes not as bounded, an earlier convergence is observed. Delayed evaluation appears to find a balance

TABLE 3. Per cent deviations from the solver solutions.

Problem	n	TW	Time	mSAC	SA	MBO	Del	1Step	Del+1Step
A	20	easy	45	149.10%	0.60%	1.60%	1.20%	3.60%	0.60%
B	20	easy	45	196.50%	11.50%	0.60%	1.90%	0.30%	0.80%
C	20	easy	45	2.70%	0.00%	0.30%	0.40%	0.60%	0.10%
D	20	easy	45	1.90%	0.30%	0.30%	1.90%	0.80%	0.50%
E	20	easy	45	152.70%	3.40%	2.70%	4.20%	3.80%	2.70%
F	20	easy	45	310.10%	12.00%	4.50%	4.40%	3.80%	1.70%
G	20	easy	45	184.10%	1.30%	1.80%	0.20%	1.10%	0.20%
			Average:	142.40%	4.10%	1.70%	2.00%	2.00%	0.90%
H1	50	easy	150	222.70%	50.60%	31.40%	29.80%	36.40%	34.60%
A	20	medium	60	84.90%	3.40%	3.20%	4.00%	2.50%	5.80%
B	20	medium	60	131.20%	0.30%	1.80%	2.30%	0.50%	0.00%
C	20	medium	60	37.40%	1.30%	7.00%	2.70%	3.30%	2.00%
D	20	medium	60	35.50%	−2.30%	−2.00%	−4.80%	−3.00%	−0.40%
E	20	medium	60	90.60%	1.90%	2.60%	5.70%	3.20%	2.10%
F	20	medium	60	99.90%	−9.00%	−6.90%	−4.80%	−7.20%	−7.40%
G	20	medium	60	98.90%	2.50%	0.10%	0.40%	0.00%	0.20%
			Average:	82.60%	−0.30%	0.80%	0.80%	−0.10%	0.30%
H1	50	medium	700	78.70%	−24.40%	−14.40%	−16.50%	−16.50%	−12.20%
H2	50	medium	700	73.70%	−24.50%	−11.80%	−14.80%	−15.40%	−13.10%
H3	50	medium	700	86.30%	−18.00%	−14.00%	−19.70%	−19.70%	−10.30%
			Average:	79.60%	−22.30%	−13.40%	−17.00%	−17.20%	−11.90%
A	20	hard	90	28.40%	−19.80%	−22.60%	−22.40%	−22.20%	−21.20%
B	20	hard	90	75.70%	1.80%	0.50%	−0.60%	0.70%	−0.10%
C	20	hard	90	21.10%	3.40%	1.00%	1.00%	1.00%	3.40%
D	20	hard	90	33.70%	2.70%	0.60%	2.90%	1.60%	1.80%
E	20	hard	90	72.90%	0.40%	0.90%	0.90%	0.10%	1.00%
F	20	hard	90	98.20%	−4.00%	−6.30%	−3.40%	−5.40%	−4.00%
G	20	hard	90	88.70%	8.20%	5.80%	8.80%	8.10%	7.00%
			Average:	59.80%	−1.10%	−2.90%	−1.80%	−2.30%	−1.70%
H1	50	hard	1200	52.70%	−27.80%	−14.90%	−19.60%	−23.20%	−23.30%
H2	50	hard	1200	29.90%	−33.10%	−31.30%	−31.50%	−33.80%	−33.30%
H3	50	hard	1200	38.50%	−31.30%	−22.40%	−27.30%	−27.40%	−29.80%
			Average:	40.30%	−30.70%	−22.90%	−26.20%	−28.10%	−28.80%

between exploration and exploitation when given limited run time. Once the complexity of the problem increases and time limitation does not become a hurdle, its solutions surpass the original MBO but converge earlier than the other variants. The situation of the one-step sharing variant fails to show dominance when given a very limited run time. Yet given an adequate time, it succeeds in showing resemblance as it does not face an early convergence. Finally, for the variant combining the two modifications of the MBO, its results show the most sluggish behavior in exploiting favored solutions. This can be disadvantageous when the time limitation is too short and an early convergence is preferred, but beneficial when observing the contrasting case as it has a wider searching ability that is essential in gigantic search spaces. This nominates the variant as the least probable one for experiencing stranding in specific search spaces the earliest.

According to the percent deviations from the lower bound and the dominance matrix, when the problem being targeted has a small number of nodes and the loosest time window, the exact solver has the most dependable performance. Such a result is expected as the exact solver does not have to create numerous variables and constraints when targeting these types of problems. In addition, the existence of a very loose time window makes meeting all constraints an easy task thus leading to an easy convergence to optimal/suboptimal solutions.

TABLE 4. Per cent deviations from the Lower Bound found.

Problem	n	TW	Time	Solver	mSAC	SA	MBO	Del	1STEP	Del+1Step
A	20	easy	45	0.00%	149.10%	0.60%	1.60%	1.20%	3.60%	0.60%
B	20	easy	45	0.00%	196.50%	11.50%	0.60%	1.90%	0.30%	0.80%
C	20	easy	45	0.00%	2.70%	0.00%	0.30%	0.40%	0.60%	0.10%
D	20	easy	45	0.00%	1.90%	0.30%	0.30%	1.90%	0.80%	0.50%
E	20	easy	45	0.00%	152.70%	3.40%	2.70%	4.20%	3.80%	2.70%
F	20	easy	45	0.00%	310.10%	12.00%	4.50%	4.40%	3.80%	1.70%
G	20	easy	45	0.00%	184.10%	1.30%	1.80%	0.20%	1.10%	0.20%
			Average:	0.00%	142.40%	4.10%	1.70%	2.00%	2.00%	0.90%
H1	50	easy	150	6.80%	244.60%	60.80%	40.30%	38.60%	45.70%	43.70%
A	20	medium	60	16.50%	115.30%	20.50%	20.20%	21.10%	19.40%	23.20%
B	20	medium	60	0.00%	131.20%	0.30%	1.80%	2.30%	0.50%	0.00%
C	20	medium	60	11.30%	53.00%	12.80%	19.10%	14.30%	15.00%	13.50%
D	20	medium	60	15.80%	56.90%	13.20%	13.50%	10.30%	12.40%	15.40%
E	20	medium	60	10.90%	111.40%	13.10%	13.80%	17.20%	14.50%	13.30%
F	20	medium	60	48.30%	196.50%	35.00%	38.10%	41.10%	37.70%	37.30%
G	20	medium	60	18.50%	135.60%	21.40%	18.60%	18.90%	18.50%	18.70%
			Average:	17.30%	114.30%	16.60%	17.90%	17.90%	16.80%	17.40%
H1	50	medium	700	123.10%	298.80%	68.60%	91.00%	86.30%	86.40%	96.00%
H2	50	medium	700	122.80%	287.00%	68.10%	96.50%	89.80%	88.60%	93.60%
H3	50	medium	700	111.80%	294.40%	73.80%	82.10%	70.00%	70.10%	90.00%
			Average:	119.20%	293.40%	70.20%	89.90%	82.00%	81.70%	93.20%
A	20	hard	90	78.30%	129.00%	42.90%	38.00%	38.40%	38.70%	40.40%
B	20	hard	90	69.10%	197.00%	72.20%	70.00%	68.10%	70.20%	68.80%
C	20	hard	90	27.40%	54.30%	31.70%	28.70%	28.70%	28.70%	31.80%
D	20	hard	90	25.90%	68.30%	29.20%	26.70%	29.60%	27.90%	28.10%
E	20	hard	90	27.60%	120.60%	28.00%	28.70%	28.70%	27.70%	28.80%
F	20	hard	90	59.70%	216.60%	53.30%	49.60%	54.20%	51.00%	53.30%
G	20	hard	90	31.60%	148.40%	42.40%	39.30%	43.30%	42.30%	40.90%
			Average:	45.60%	133.40%	42.80%	40.10%	41.60%	40.90%	41.70%
H1	50	hard	1200	177.80%	324.30%	100.70%	136.50%	123.40%	113.40%	113.00%
H2	50	hard	1200	249.30%	353.60%	133.80%	140.00%	139.10%	131.20%	133.00%
H3	50	hard	1200	194.70%	308.10%	102.60%	128.60%	114.20%	114.00%	106.80%
			Average:	207.30%	328.70%	112.40%	135.00%	125.60%	119.50%	117.60%

TABLE 5. Dominance matrix.

Node size	Best Method		
20	Solver	SA	MBO
50	Solver	SA	SA
	easy	medium	hard
	TW Difficulty		

When the number of nodes is still small, but the time window gets tighter, the solver struggles to find quality solutions while simultaneously satisfying all the constraints. Some of the matheuristics here can outperform as they succeed in finding solutions of better quality than the ones provided from the solver, given the same runtime.

When switching to nodes with higher dimensions and a loose time window, the exact solver dominates as one of the problem's main aspects is not present (time windows that must be satisfied). This is a major setback for metaheuristics as their framework depends on working cooperatively with an exact solver. Each time one of these

algorithms finds a new visiting sequence, it feeds it to an exact solver to optimize the mode of transportation between the node pairs and the entry time of each node. This time-consuming process is not needed in problems with a very loose time window as one major subproblem of the whole problem is almost no longer needed. The time every node is entered is not crucial anymore because satisfying it is an easy task to achieve. This defeats the purpose of including an exact solver within the framework of the metaheuristics as it hinders their progress and results in them failing to provide solutions of quality in a limited amount of time.

Subsequently, when the problem as a whole starts to get targeted with the inclusion of time windows that get tighter, the utilization of the metaheuristics becomes undeniable. At that point, when talking about average deviations from the lower bound as shown in Table 4, all the matheuristics start to drastically over-perform the exact solver's solution.

As a result, the quantitative study suggests the use of an exact solver for problems with nodes having no or very loose time windows. When the problem has a small number of nodes but very narrow time windows, the use of the original version of MBO is recommended. For problem types with small node sizes and medium time window tightness, and for problems with large node sizes and medium or hard time window, the use of SA is advised. However, if the user has enough time, obviously trying all methods and picking the best one would be the best strategy.

5. Summary and conclusions

This paper presents a comprehensive study of a new variant of the Turkish cashier problem (TCP) that incorporates the factor of time, which is referred to as the Turkish cashier problem with time windows (TCPwTW). The primary objective of this study was to investigate and provide solutions for this variant, while also examining the complexity involved in solving such a problem. To accomplish this, the existing literature on methodologies targeting similar or identical problems was reviewed.

A detailed description of the problem is provided to enhance understanding of its structure and the how time aspect was integrated into the original TCP. Subsequently, various methodologies with modifications were discussed as potential approaches to address this complex problem. Among these methodologies, a heuristic algorithm called modified SAC (mSAC) was proposed, along with matheuristics (metaheuristics) that employed an exact solver in their iterations. The metaheuristics employed in this study included simulated annealing and four variants of the MBO. Furthermore, studies were conducted to identify the most effective neighborhood search methods associated with the metaheuristics. Additionally, the process of approximating the optimal parameters for each metaheuristic, based on the specific characteristics of the problem at hand, was addressed.

Finally, all the proposed methodologies were implemented and compared alongside the best solution attainable from the exact solver within a given time limit. This quantitative study allowed for a comprehensive evaluation of the outcomes obtained by the exact solver and the alternative methods, based on their deviation from the lower bound (LB) reported by the exact solver.

In conclusion, this research highlights the significance of studying such a problem based on its various applications that can be faced in real life. In addition to selecting appropriate methodologies based on the complexity of the settings associated with given problems. Through a quantitative study that compares the results obtained from an exact solver and alternative heuristic methods, this paper offers valuable insights into choosing the most effective approaches for solving the Turkish cashier problem, considering its unique time sensitive variant and varying problem settings that can be faced.

The problem mentioned can be applicable in more scenarios faced in practice. For large companies, such a problem might not be fruitful as affording to provide car is not an issue. Although, when considering individuals without a car because of monetary issues, or the inability to drive due to legal restrictions regarding an age requirement, the problem becomes more encountered. Take students (high school students or lower levels) as an example, for the most part, they neither have the money to own a private car nor possess the legal requirement to drive.

As observed from Section 4, the most powerful methods to adapt when attacking problems possessing high complexity was the metaheuristics merged with the use of an exact solver through the iterations(matheuristics).

Feeding newly generated solutions to an exact solver to obtain a full solution is a process that costs the metaheuristics a valuable amount of their run time thus limits the exploration areas in the search space. If the time it takes to operate each iteration is shortened, more time can be spared, and this additional time could have been invested in exploring extended areas of the neighborhood space.

As a further study, the formulation of a methodology that can generate the three elements of a complete solution without the use of an exact solver in an efficient and correlated way can open the possibility of stumbling upon solutions that surpass the ones obtained using our methods.

Data availability statement

The data generated and used in this paper is made publicly available and can be accessed through the following Github repository: (https://github.com/AhmadBassaleh/TCPwTWData) [2].

References

[1] A. Agrawal, N. Ghune, S. Prakash and M. Ramteke, Evolutionary algorithm hybridized with local search and intelligent seeding for solving multi-objective Euclidian TSP. *Expert Syst. Appl.* **181** (2021) 115192.

[2] A. Bassaleh and E. Duman, Java code for "Data for the Turkish Cashier Problem with Time Windows (TCPwTW)" (2023). https://github.com/AhmadBassaleh/TCPwTWData.

[3] E. Duman, A new application of the traveling salesman problem: the turkish cashier problem. *Comput. Appl. Math.* **21** (2022) 259-268.

[4] E. Duman and I. Or, Precedence constrained TSP arising in printed circuit board assembly. *Int. J. Prod. Res.* **42** (2004) 67-78.

[5] T. Duman and E. Duman, Solving a new application of asymmetric TSP by modified migrating birds optimization algorithm. *Evol. Intell.* (2023).

[6] X. Geng, Z. Chen, W. Yang, D. Shi and K. Zhao, Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Appl. Soft Comput.* **11** (2011) 3680-3689.

[7] F. Glover, G. Gutin, A. Yeo and A. Zverovich, Construction heuristics for the asymmetric TSP. *Eur. J. Oper. Res.* **129** (2001) 555-568.

[8] H. Hashimoto, T. Ibaraki, S. Imahori and M. Yagiura, The vehicle routing problem with flexible time windows and traveling times. *Discrete Appl. Math.* **154** (2006) 2271-2290.

[9] X. He, Q.K. Pan, L. Gao and J. Neufeld, An asymmetric traveling salesman problem based matheuristic algorithm for flowshop group scheduling problem. *Eur. J. Oper. Res.* (2023).

[10] M.Junger, G. Reinelt and G. Rinaldi, The traveling salesman problem. *Handbooks Oper. Res. Manag. Sci.* **7** (1995) 225-330.

[11] H. Kona, A. Burde and D. Zanwar, A review of traveling salesman problem with time window constraint. *IJIRST-Int. J. Innov. Res. Sci. Eng. Technol.* **2** (2015).

[12] R. Lahyani, M. Khemakhem and F. Semet, A unified matheuristic for solving multi-constrained traveling salesman problems with profits. *EURO J. Comput. Optim.* **5** (2017) 393-422.

[13] J. Lenstra and A. Kan, Complexity of vehicle routing and scheduling problems. *Networks* **11** (1981) 221-227.

[14] T. Rao, A comparative evaluation of GA and SA TSP in a supply chain network. *Mater. Today: Proc.* **4** (2017) 2263-2268.

[15] M. Shahmanzari, D. Aksen and S. Salhi, Formulation and a two-phase matheuristic for the roaming salesman problem: Application to election logistics. *Eur. J. Oper. Res.* **280** (2020) 656-670.

[16] X. Shi, Y. Liang, H. Lee, C. Lu and Q. Wang, Particle swarm optimization-based algorithms for TSP and generalized TSP. *Inf. Process. Lett.* **103** (2007) 169-176.

[17] R. Silva, E. Venites Filho and A. Alves, A thorough study of the performance of simulated annealing in the traveling salesman problem under correlated and long tailed spatial scenarios. *Phys. A: Stat. Mech. Appl.* **577** (2021) 126067.

[18] R. Skinderowicz, Improving Ant Colony Optimization efficiency for solving large TSP instances. *Appl. Soft Comput.* **120** (2022) 108653.

[19] M. Solomon and J. Desrosiers, Survey Paper–Time Window Constrained Routing and Scheduling Problems. *Transp. Sci.* **22** (1988) 1–13.

[20] T. Srinivas Rao, A simulated annealing approach to solve a multi traveling salesman problem in a FMCG company. *Mater. Today: Proc.* **46** (2021) 4971–4974.

[21] A. Subramanian and M. Battarra, An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. *J. Oper. Res. Soc.* **64** (2013) 402-409.

[22] İ Tarhan and C. Oğuz, A matheuristic for the generalized order acceptance and scheduling problem. *Eur. J. Oper. Res.* **299** (2022) 87-103.

[23] V. Tongur and E. Ülker, The analysis of migrating birds optimization algorithm with neighborhood operator on traveling salesman problem. In: *Intelligent and Evolutionary Systems: The 19th Asia Pacific Symposium, IES 2015, Bangkok, Thailand, November 2015, Proceedings* (2016) 227-237.

[24] S. Tonyali and A. Alkaya, Application of recently proposed metaheuristics to the sequence dependent TSP. In: *Advanced Computational Methods for Knowledge Engineering: Proceedings of 3rd International Conference on Computer Science, Applied Mathematics and Applications-ICCSAMA 2015* (2015) 83-94.

[25] M. Uwaisy, Z. Baizal and M. Reditya, Recommendation of scheduling tourism routes using tabu search method (case study bandung). *Procedia Comput. Sci.* **157** (2019) 150-159.

Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at https://edpsciences.org/en/subscribe-to-open-s2o.