INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2025) 1–30
DOI: 10.1111/itor.70125

A matheuristic for the traveling salesman problem with positional consistency constraints

Luís Gouveia[a,c], Ana Paias[a,c] and Mafalda Ponte[b,c,*]

[a] *DCM, Faculdade de Ciências, Universidade de Lisboa, C6, Piso 4, Lisboa 1749-016, Portugal*
[b] *DMQGE, ISCTE – Instituto Universitário de Lisboa, Edifício 2, Piso 4, Lisboa 1649-026, Portugal*
[c] *Center for Mathematical Studies, University of Lisbon (CEMS.UL), Faculdade de Ciências, Universidade de Lisboa, Lisboa 1749-016, Portugal*
E-mail: legouveia@ciencias.ulisboa.pt[Gouveia]; ampaias@ciencias.ulisboa.pt[Paias]; mafalda.ponte@iscte-iul.pt[Ponte]

Received 7 March 2025; received in revised form 29 October 2025; accepted 29 October 2025

Abstract

We propose a matheuristic for the traveling salesman problem with positional consistency constraints, where we seek to generate a set of routes with minimum total cost, in which the nodes visited in more than one route (consistent nodes) must occupy the same relative position in all routes. The matheuristic is an iterated local search based algorithm that uses a restricted version of the problem under study, where the positions of consistent nodes are fixed, to significantly improve the quality of local optima found by the local search. Computational results show that, for instances with 48–171 nodes and 5 or 10 routes, the matheuristic can obtain, in short computational times, significantly better solutions than an exact method in 10 hours, obtaining optimal or near-optimal solutions for instances where the optimal solution is known.

Keywords: combinatorial optimization; traveling salesman problem; positional consistency; iterated local search

1. Introduction

COVID-19 lockdowns brought additional pressure on national healthcare systems. At that time, while addressing urgent pandemic-related needs, it was also essential to maintain regular healthcare services for the general population. Additionally, minimizing patients' time in health centers was crucial to reduce the risk of contagion, so the goal was to create cost-efficient schedules for teams of healthcare professionals working at the same health center. The working day is divided into equal-length time slots (e.g., 30 minutes each), during which various tasks must be assigned. These primarily involve attending to patients, but occasionally include team meetings. Meetings

*Corresponding author.

may be limited to doctors, nurses, or involve the entire team. Patient appointments also vary in professional requirements: some require only a doctor (e.g., family planning), others only a nurse (e.g., vaccinations), and some require both (e.g., pediatric care). When a patient needs multiple professionals, they must be scheduled for the same time slot to avoid extended stays at the facility. The efficiency of a schedule can be assessed using the dissimilarity of tasks that are performed consecutively, in the sense that it is more efficient to transition from a task to another one that uses similar equipment, in opposition to another one that would take place in a different room. This can be modeled as a routing problem, using a graph where each node represents a task assigned to a healthcare professional, each route corresponds to a professional's work shift, and relative positions within the route represent time slots in that shift. Consistency must be enforced at the time-slot level, meaning that tasks must align based on their relative positions.

In the context of multiple route optimization problems, synchronization can be described as an interdependence between the different routes. One of the most fundamental examples of this interdependence can be seen in the fact that in most applications customer nodes are visited by one and only one route, and therefore the assignment of a client to a route results in the other routes not being allowed to visit it. Another example of synchronization appears in problems that allow transshipment. The amount to be transshipped depends on the capacity of the vehicle that receives the load and the quantity of goods still available in both vehicles. In addition, the problem might include the existence of limited resources to be shared among the different vehicles. One such example can be found in Nolz et al. (2022), where vehicles are electric and there is a limited number of charging stations, thus imposing an upper limit for the number of vehicles recharging simultaneously. A review of vehicle routing problems (VRPs) with synchronization constraints was carried out by Drexl (2012).

Consistency is a type of synchronization and can take many forms. Kovacs et al. (2014a) provide a survey on the different types of consistency constraints, applications where such constraints are relevant, and methodologies used to address the consistent traveling salesman problem (TSP) and the consistent VRP. The authors consider three main different types of consistency: *delivery consistency*, meaning that clients with several deliveries in the planning horizon receive roughly the same amount of product in all deliveries, or that deliveries are regularly spaced within the planning horizon; *vehicle consistency*, meaning that a customer will always be visited by the same vehicle; and *service consistency*, meaning that a customer requiring several visits must be visited roughly at the same time. Service consistency applies when the same customer must be visited in several time periods or when the customer must be visited by several vehicles in the same time period. In general, consistency is desirable to improve the customer experience and efficiency. In fact, delivery consistency results in a more predictable service, improving customer satisfaction; vehicle consistency results in familiarity between the customer and the driver, which is beneficial both for efficiency (more predictable routes for the driver) and customer satisfaction; and service consistency results in a more personalized and predictable service. Consistency may also be a prerequisite in several applications. For example, in the problem studied by Anderluh et al. (2017), where two types of vehicles are considered (bicycles and vans), the transshipping points have no storage capacity and therefore bicycles can receive only more goods to deliver if both the bicycle and the van are at the transshipment point at the same time. This work will focus on service consistency, following up on the TSP with positional consistency constraints (CTSP), as described by Gouveia et al. (2023).

After choosing the type(s) of consistency to be incorporated in a formulation, several decisions must be made regarding its enforcement.

Incorporating time. Because service consistency means that several vehicles must visit the same client (roughly) at the same time, it is necessary to incorporate time into a formulation. The most common approach is through a weight matrix, with the weights representing travel times, while defining variables to represent the starting times for the tasks/visits (*temporal consistency*). Although most of the works available in the literature focus on temporal consistency, Gouveia et al. (2023) proposed an approach based on *positional consistency*, which is adequate if the transition times are small or very similar and the service times are the same for all nodes. This approach consists of dividing the working day into homogeneous time slots and using the relative position of a node in a route to represent the allocation of that node to a time slot. In this case, the relative positions are used to model consistency, and the authors showed that time-dependent formulations (see, e.g., Picard and Queyranne, 1978) with consistency constraints that took advantage of their position index were the most efficient to address the problem.

Measuring consistency. The two most common measures of temporal consistency are *time slacks* (Groër et al., 2009) and *time classes* (Feillet et al., 2014). The former consists of the difference between the latest and the earliest arrival times for the same node requiring consistency. The smaller the slack, the *more consistent* the routes are. The latter generates, for each node, the minimum number of time classes needed to cover all of the arrival times for the node, with two arrival times only being allowed to belong to the same time class if their difference does not exceed a maximum bound that is given as an input. The routes are considered the more consistent the fewer the time classes needed to cover the arrival times for the consistent nodes. Note that time classes are not equivalent to the time slots in the positional consistency approach proposed by Gouveia et al. (2023). Time classes are associated with transition times (and service times, if applicable), whereas positional consistency makes use of the relative position of the nodes in the routes. Only consistent nodes need time classes, and more than one node from the same route may appear in the same time class if transition times are small enough. However, all nodes are assigned time slots, and two nodes on the same route cannot occupy the same time slot. Time classes are suitable when travel/service times are relevant, and services may start at any moment in the planning horizon. Time slots start and end at specific times, given as input, since they correspond to specific time intervals in a schedule. Time classes may not cover the entire planning horizon, since they are generated as necessary, and there may be moments of the planning horizon that are covered by several time classes. Time slots, on the other hand, cover the whole planning horizon and do not intersect.

Hard or soft constraints. The way consistency is enforced depends on whether consistency is merely a means to improve customer satisfaction or a prerequisite for the service. In the first case, it is possible to consider consistency in the objective function, or in a multiobjective approach, by penalizing differences in arrival times or deviations from ideal service times/soft time windows, or even by minimizing the number of different time classes the same client is assigned to. In the second case, consistency can be enforced through the definition of hard time windows, to be respected by all of the vehicles that participate in the same task, or constraints imposing a limit to the time slack or the number of different time classes that the same client can be served in. In extreme, we may aim for *total consistency*, where a task is performed exactly at the same time by all the vehicles involved in it. In the context of temporal consistency, this may be achieved by setting the time slack to zero, which, however, may not be practical, especially if idle times are not allowed. On the other hand,

total consistency is adequate in positional consistency, being modeled by constraints that state that a task must be performed in the same relative position in all of the routes it appears in.

Idle times. Idle/waiting times can be incorporated into a formulation to obtain cheaper feasible solutions or even to ensure that it is possible to obtain at least one feasible solution in applications where consistency constraints or hard time windows must be enforced. Several approaches regarding waiting times can be found in the literature for routing problems with consistency constraints: (i) not allowing waiting times (Feillet et al., 2014); (ii) allowing flexible departure times, but no waiting times in the middle of a route (Kovacs et al., 2015b); (iii) allowing waiting times in the middle of a route, but only when needed to enforce time windows (Braekers et al., 2016); (iv) fully allowing waiting times (Ioachim et al., 1999). Considering the application that inspired the CTSP, unnecessary permanence of the health practitioners in the facility was heavily discouraged and, therefore, idle positions/times are not allowed. Starting positions are flexible, in the sense that different practitioners can start their work shifts at different times. However, if a shift (route) 1 starts one hour after the facility opens, and each time slot is 30 minutes, then the first time slot of shift 1 corresponds to the third time slot of a shift, 2, which began at opening time. To maintain consistency across different shift start times, all time slots in later-starting routes must be normalized by adjusting for the delay.

Sequential consistency. Sequential consistency states that if clients i and j require consistency, they must be visited in the same order in all of the routes they appear simultaneously in. Several approaches, with or without sequential consistency, can be found in the literature for routing problems with consistency constraints. For example, the constraints used by Subramanyam and Gounaris (2016) to enforce temporal consistency indicate that it is possible to have two paths containing two nodes, i and j, one starting in i and ending in j in one route, and one starting in j and ending in i in another route, as long as the differences between arrival times do not exceed a given limit. However, the enforcement of sequential consistency is a prerequisite in all heuristics using the so-called *template approach* (see, e.g., Groër et al., 2009; Kovacs et al., 2014b), which will be relevant for the methodologies presented in this work. Sequential consistency is implied by total consistency.

Table 1 summarizes several works available in the literature, which address routing and/or scheduling problems with service consistency. For each paper, the table indicates whether total consistency and idle times are allowed, how consistency is enforced, and the main methodologies proposed for the problem. Regarding the latter, it can be seen that, due to the complexity of problems with consistency constraints, the majority of the methodologies proposed to address them are metaheuristics and matheuristics, specifically local search based heuristics. Following this trend, and to address larger instances than the ones solved by Gouveia et al. (2023), this study proposes an iterated local search (ILS) based matheuristic for the TSP with positional consistency.

The contributions of this paper are as follows: (i) We define a restricted version of the CTSP, where the positions of the consistent nodes are known and fixed, and free nodes and empty positions are allowed to vary, which can be solved to optimality in few seconds, even for relatively large instances. (ii) We present an ILS-based matheuristic that uses the restricted version of the CTSP to significantly improve the quality of local optima found by the local search. (iii) We carry out a computational experiment to assess the performance of the matheuristic, comparing it with the performance of an ILS algorithm that uses the same constructive heuristic, local search algorithm, and perturbation as the matheuristic, and also comparing the solutions from the matheuristic with the best solutions obtained from an exact method after 10 hours. (iv) We present a new set of

Table 1
Routing and scheduling with consistency in the literature

Paper	Total Consistency	Idle Times	Consistency enforcement	Methodology
Anderluh et al. (2017)	✓	Allowed	Consistency constraints (time slack)	Two-phase heuristic
Andersson et al. (2011)		Allowed	Consistency constraints (time slack)	Column generation
Braekers et al. (2016)		For time windows	Time windows and multiple objectives	Multidirectional local search and large neighborhood search
Bredström and Rönnqvist (2008)	✓	Allowed	Consistency constraints (time slack)	Branch-and-bound and hybrid heuristic
Cappanera et al. (2020)		Not allowed	Consistency constraints (time slack)	MILP formulations and lower bounding
Ceschia et al. (2026)	✓	For time windows	Consistency constraints (time slack)	Multineighborhood simulated annealing
Decerle et al. (2017)		Allowed	Multiple objectives (time slack)	Memetic algorithm
Emadikhiav et al. (2020)Emadikhiav, Bergman and Day)		Allowed	Consistency constraints (time slack)	Branch and check/branch and price
En-nahli et al. (2016)	✓	Allowed	Consistency constraints (time slack)	Iterated local search
Feillet et al. (2014)		Not allowed	Multiple objectives (time classes)	Large neighborhood search
Fink et al. (2019)	✓	Allowed	Consistency constraints (time slack)	Column generation
Frifita and Masmoudi (2020)	✓	For time windows	Consistency constraints (time slack)	MILP formulation and variable neighborhood search
Goeke et al. (2019)		Not allowed	Consistency constraints (time slack)	Column generation
Large neighborhood search				
Gouveia et al. (2023)	✓	Flexible departure	Consistency constraints (positional)	Branch and cut
Groër et al. (2009)	Not allowed	Consistency constraints (time slack)	Record-to-record travel algorithm	

Continued

Table 1
(Continued)

Paper	Total Consistency	Idle Times	Consistency enforcement	Methodology
Ioachim et al. (1999)	✓	Allowed	Consistency constraints (time windows)	Decomposition method and column generation
Kovacs et al. (2014b)	Flexible departure	Consistency constraints (time slack)	Large neighborhood search	
Kovacs et al. (2015b)		Flexible departure	Multiple objectives (time slack)	Optimize each objective individually (bounding the others)
	Multidirectional local search and large neighborhood search			
Kummer et al. (2024)	✓	For time windows	Consistency constraints (time slack)	Biased random-key genetic algorithm
Lespay and Suchan (2021)		For time windows	Time windows	Multiperiod insertion heuristic
Lian et al. (2016)		Not allowed	Multiple objectives (time slack)	Multidirectional local search
Liu et al. (2021)	✓	For time windows	Consistency constraints (time slack)	Memetic algorithm
	Hybrid genetic and variable neighborhood search			
	Hybrid genetic and simulated annealing			
	Hybrid simulated annealing			
Luo et al. (2015)		Allowed	Time windows	Heuristic decomposition method and tabu search
Mancini et al. (2021)		Allowed	Consistency constraints (time slack)	ILS-based matheuristic
Mankowska et al. (2014)	✓	For time windows	Consistency constraints (time slack)	ILP formulation and adaptive variable neighborhood search
Nolz et al. (2022)		Allowed	Multiple objectives (time slack)	Adaptive large neighborhood search
Parragh and Doerner (2018)	✓	Allowed	Consistency constraints (time slack)	Adaptive large neighborhood search

Continued

Table 1
(Continued)

Paper	Total Consistency	Idle Times	Consistency enforcement	Methodology
Stavropoulou et al. (2019)		Allowed	Consistency constraints (time slack)	Adaptive tabu search
Subramanyam and Gounaris (2016)		Not allowed	Consistency constraints (time slack)	Branch and cut
Subramanyam and Gounaris (2018)		Allowed	Consistency constraints (time slack)	Decomposition method
Tarantilis et al. (2012)		Not allowed	Consistency constraints (time slack)	Tabu search
Tellez et al. (2022)		For time windows	Multiple objectives (time classes)	Solving set partitioning problem and large neighborhood search
Wang et al. (2022)		Flexible departure	Consistency constraints (time slack)	Column-and-cut generation
Xu and Cai (2018)		Not allowed	Consistency constraints (time slack)	Two-stage variable neighborhood search

ILP, integer linear programming; ILS, iterated local search; MILP, mixed-integer linear programming.

relatively large-sized test instances for the CTSP, ranging from 48 to 171 nodes, with 5 and 10 routes, which are available online.

Section 2 provides a formal definition for the TSP with positional consistency constraints. Section 3 is focused on the restricted CTSP (rCTSP) and provides the integer linear programming (ILP) formulation that was incorporated into the matheuristic. Section 4 provides details on the matheuristic, namely how the initial solution is obtained, how the local search is performed, which perturbation is applied to local optima, and how the ILP formulation is incorporated into the algorithm, followed by Section 5, which details the computational experiment carried out to evaluate the performance of the algorithm. Section 6 presents the main conclusions of this study, along with possible future research lines.

2. The traveling salesman problem with positional consistency

The TSP with total positional consistency (CTSP) is defined in a complete graph, $G = (V, A)$, where V is the set of nodes, $n = |V|$, and A is the set of arcs. Node 1 is the depot (representing inactivity), whereas the subset $V \setminus \{1\}$ defines the set of client nodes (the tasks for the health practitioners), which is divided into nondisjoint subsets m, V_l, $l = 1, \ldots, m$. For each arc (i, j), the cost of traveling from node i to node j, C_{ij}, is known. The purpose of the problem is to define a minimum cost set of m routes, each route l starting and finishing at the depot while visiting all the client nodes in V^l, each node occupying one and only one relative position. Clients served on more than one route require total positional consistency. A node requiring total positional consistency is visited in the same relative position in all routes in which it is included.

Different routes can have different sizes, which must be taken into account in the definition of consistency. For simplicity, we will assume that there are as many available positions as client nodes in the largest route (T). Smaller routes are allowed to start "later" and/or finish "earlier" than the largest route, as long as they start in position 1 or later, finish in position T or earlier, and do not have empty positions between positions that are occupied by nodes. Figure 1 uses a small example to illustrate the need to allow flexible start times. Consider two healthcare professionals who, for the next work shift, share two tasks. Practitioner H1 was assigned tasks t1, t2, t3, t4, t5, and t6, while practitioner H2 was assigned tasks t1, t2, t7, and t8. Since task t1 is common to both practitioners, it requires consistency and must be assigned to the same time slot by both. The same is true for task t2. Figure 1c shows a graph representation of a solution for the CTSP. The natural interpretation of the solution is to attribute to each task the relative position it occupies on the route, as shown in solution S1. This is not a feasible solution because consistent nodes occupy different time slots for different practitioners. For example, t2 occupies the time slot TS2 for practitioner H1, but is the first task in the route relating to H2. However, it is possible to obtain a feasible solution by allowing the route of H2 to start later, as shown in solution S2. In the application to healthcare services, this corresponds to having (if, for instance, each time slot consists of 30 minutes) practitioner H1 start working as soon as the facility opens at 8:00 a.m., while practitioner H2 starts working at 8:30 a.m.

Therefore, we define *local position* as the relative position that a node occupies on a route, when that route is taken independently of the others. The *global position* is the position that a node occupies on a route when taking into account whether that route starts in the same position as the largest route. In fact, for a route that starts later, the node in the first local position may occupy the

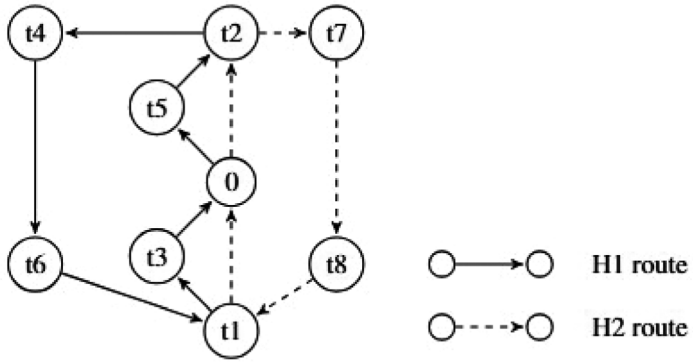
International Transactions in Operational Research published by John Wiley & Sons Ltd on behalf of International Federation of Operational Research Societies.

	TS1	TS2	TS3	TS4	TS5	TS6
H1	t5	t2	t4	t6	t1	t3
H2	t2	t7	t8	t1	-	-

(a) S1

	TS1	TS2	TS3	TS4	TS5	TS6
H1	t5	t2	t4	t6	t1	t3
H2	-	t2	t7	t8	t1	-

(b) S2



(c) Graph representation of S1 and S2

Fig. 1. Solution S1 is not feasible for the problem, since tasks t1 and t2 require consistency but have been assigned to different time slots. However, a feasible solution S2 can be obtained from these sequences of tasks by allowing practitioner H2 to start working in time slot 2. Both solutions are represented by the same set of routes.

$$\begin{bmatrix} 6 & 3 & 2 & 4 & 5 \\ 3 & 2 & 7 & 8 & 0 \\ 0 & 6 & 9 & 2 & 4 \end{bmatrix}$$

(a) Solution 1

$$\begin{bmatrix} 6 & 3 & 2 & 4 & 5 \\ 0 & 3 & 2 & 7 & 8 \\ 6 & 9 & 2 & 4 & 0 \end{bmatrix}$$

(b) Solution 2

Fig. 2. Solutions for the CTSP are represented using solution matrices, where each line represents a route and each column represents a position. Entries with value 0 indicate positions that are empty in a route. Solution 1 is not feasible because nodes 2, 3, 4, and 6 are visited in several routes but do not satisfy consistency. Comparing route 3 with route 1, the consistent nodes occupy the same local position, but not the same global position, because route 3 starts later than route 1. In route 2, the consistent nodes do not occupy the same position they occupy in other routes (neither local nor global). It is possible to obtain a feasible solution by maintaining the sequence of nodes in each route while making route 3 start earlier and allowing route 2 to start later, as illustrated in Solution 2.

kth ($k > 1$) global position, as illustrated in Fig. 2. Consistency constraints will be written in terms of global positions, so hereinafter, a node's global position will be referred to simply as "position."

As shown in Fig. 2, CTSP solutions are represented using a matrix $m \times T$, $msol$, where each line represents a route and each column represents a global position. For a given k and l, an entry $msol(l, k)$ can take value 0, indicating that position k is empty on route l, or $i \in V^l$, which means that the client node i occupies position k on route l. Since idle times are not allowed per the application, entries with a 0 may appear at the beginning and/or end of a route, but not between entries occupied by client nodes; that is, the positions occupied by the nodes are consecutive.

If we remove the consistency constraints, the CTSP reduces to m TSPs, one for each period. Because the TSP is an NP-hard problem, we can conclude that the CTSP is also NP-hard.

Several formulations were proposed by Gouveia et al. (2023) for the CTSP. The computational experiments reported in that article indicate that the two most efficient formulations to solve CTSP are the Picard and Queyranne formulation (*PQ*), an adaptation of a formulation proposed by Picard and Queyranne (1978) with additional consistency constraints, and the Aggregated Picard and Queyranne formulation (*APQ*), a new formulation that can be viewed as an aggregation of the PQ formulation, with the *APQ* model outperforming the *PQ* model in many cases. However, even for these more efficient formulations, instances with 5 routes and 44 nodes proved to be challenging, with some of them not being solved to optimality by any of the formulations within a time limit of three hours.

To address larger instances, both in terms of the number of nodes and the number of routes, we propose a matheuristic for the CTSP, which has the usual structure of an ILS algorithm, but, in some iterations, solves an rCTSP induced by the final solution of the local search.

The optimal solution for the rCTSP is feasible for the original CTSP. The main reason for using this restricted version in the ILS algorithm is that larger instances can be solved in short computational times. In fact, instances for the rCTSP with up to 76 nodes distributed among 5 routes, as well as with up to 170 nodes distributed among 10 routes, can be solved to optimality in a few seconds. The rCTSP is described in the next section.

3. The restricted CTSP

The rCTSP is a restricted version of the CTSP where the positions of the consistent nodes are known (and therefore fixed) *a priori*, whereas the positions of the free nodes and the empty positions (if applicable) are allowed to vary, as long as two nodes from the same route do not occupy the same position and there are no empty positions in the middle of a route. For a more clear explanation of the rCTSP, we now present the ILP formulation that was selected and incorporated into the hybrid ILS heuristic:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{T}\sum_{l=1}^{m} C_{ij} z_{ij}^{kl} \tag{1}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{T} z_{ij}^{kl} = S_j^l;\ \forall j = 1,\ldots,n;\ \forall l = 1,\ldots,m \tag{2}$$

$$\sum_{i=1}^{n} z_{ij}^{kl} = \sum_{i=1}^{n} z_{ji}^{k+1,l};\ \forall j = 2,\ldots,n;\ \forall k = 1,\ldots,T-1;\ \forall l = 1,\ldots,m \tag{3}$$

$$p_i^{posi} = 1, \forall i \in V^{cons} \tag{4}$$

$$\sum_{k=1}^{T-1} p_j^k = 1;\ \forall j = 2,\ldots,n \tag{5}$$

$$\sum_{j=2}^{n} S_j^l p_j^k \leq 1;\ \forall l = 1,\ldots,m;\ \forall k = 1,\ldots,T-1 \tag{6}$$

$$S_j^l p_j^k = \sum_{i=1}^{n} z_{ij}^{kl};\ \forall j = 2, \ldots, n;\ \forall k = 1, \ldots, T-1;\ \forall l = 1, \ldots, m \tag{7}$$

$$z_{ij}^{1l} = 0;\ \forall i = 2, \ldots, n;\ \forall j = 1, \ldots, n;\ \forall l = 1, \ldots, m \tag{8}$$

$$z_{ij}^{Tl} = 0;\ \forall i = 1, \ldots, n;\ \forall j = 2, \ldots, n;\ \forall l = 1, \ldots, m \tag{9}$$

$$z_{1j}^{kl} = 0, \forall j = 2, \ldots, n, l = 1, \ldots, m, k = T - N^l + 2, \ldots, T \tag{10}$$

$$z_{i1}^{kl} = 0, \forall i = 2, \ldots, n, l = 1, \ldots, m, k = 1, \ldots, N^l - 1 \tag{11}$$

$$p_i^k \in \{0;\ 1\};\ \forall i = 2, \ldots, n;\ \forall k = 1, \ldots, T-1 \tag{12}$$

$$z_{ij}^{kl} \in \{0; 1\};\ \forall i = 1, \ldots, n;\ \forall j = 1, \ldots, n;\ \forall k = 1, \ldots, T;\ \forall l = 1, \ldots, m. \tag{13}$$

This formulation is an adaptation of one of the variants of the PQ formulation, presented in Gouveia et al. (2023). The formulation uses binary variables z_{ij}^{kl}, which take the value 1 if the arc (i, j) occupies position k in route l, 0 otherwise, and p_i^k, $i = 2, \ldots, n, k = 1, \ldots, T-1$, which take the value 1 if node i occupies position k in the solution and 0 otherwise. The objective function (1) states that the total cost of the set of routes must be minimized. Also, if node j must be visited on route l, then on that route there must be an arc entering j, as stated by constraints (2). Flow conservation constraints (3) state that if there is an arc entering node j at position k in route l, then there must be an arc leaving j at position $k+1$ on that same route. Constraints (4) allow us to fix the positions of consistent nodes. If V^{cons} is the set of consistent nodes, $i \in V^{cons}$ and *posi* is the position that node i occupies in the input solution, then i must occupy position *posi* (in all the routes it is visited in). Consistency is modeled through constraints (5)–(7): The first set states that each client node occupies one and only one position, the second set states that two nodes that are visited on the same route cannot occupy the same position, and the third set links the two sets of decision variables, stating that if there is an arc entering node j in position k in route l, then j occupies position k. Constraints (8) indicate that an arc in position 1 cannot leave from a node other than the depot, while Equations (9) state that an arc in position T can enter only the depot. Constraints (10) and (11) further remove arcs leaving the depot too "late" and arcs entering the depot too "early" for it to be possible for all nodes of the route to occupy consecutive positions. These constraints are not necessary for the model to be valid, but result in a stronger LP relaxation. Finally, the decision variables are binary (constraints (12) and (13)).

4. The iterated local search based matheuristic

The matheuristic proposed for the CTSP is an ILS-based algorithm. The ILS approach iteratively applies local search to build sequences of solutions, where the first solution of a sequence is obtained by applying a perturbation to the last solution of a previous sequence. Four main modules define this methodology: the *initial solution*, the *local search algorithm*, the *perturbation*, and the *acceptance criterion*. The initial solution is usually obtained through a constructive heuristic and constitutes the starting point for the ILS algorithm. The second module, referred to as local search for simplicity, is the improvement heuristic applied to the initial solution, to try and obtain a better

solution, and does not necessarily have to be a simple local search algorithm (alternatives such as tabu search and simulated annealing, for instance, would also be valid). To escape local optima, the initial solution for a new sequence is obtained by applying a transformation (*perturbation*) to the last solution of a previous sequence, guaranteeing, through randomization, that diversity in the search is maintained. Finally, the acceptance criterion is the rule used to decide which solution a perturbation should be applied to in each iteration of the algorithm, which also regulates diversity in the search. The extreme case of diversification occurs when the perturbation is always applied to the last solution of the current iteration, regardless of its cost. The (opposite) extreme case of intensification occurs when the perturbation is always applied to the best solution found so far in the search. An in-depth explanation of this metaheuristic is given, for example, in Lourenço et al. (2003).

These four modules of the ILS algorithm, as well as the way the ILP formulation for the rCTSP is incorporated into the matheuristic, are detailed in Sections 4.1–4.4, followed by an overview of the whole algorithm, in Section 4.5.

4.1. Initial solution

This section explains how the initial solution is obtained for the ILS. A simple procedure, inspired by the template approach proposed by Groër et al. (2009), was developed to generate feasible solutions for the CTSP. The template approach uses permutations of nodes, referred to as the *template*, to indicate the order of the nodes in the routes. In the original approach, templates include only consistent nodes. To obtain a route from a template, the consistent nodes not visited on the route are removed, and the free nodes are heuristically inserted. However, subsequent versions of the methodology (see, for instance, Xu and Cai, 2018) were proposed using template vectors with all nodes. In this case, a route is obtained by removing from the template all the nodes that are not visited in it. The constructive heuristic proposed for the CTSP, detailed in Algorithm 1, follows these latter approaches, by considering a permutation of all client nodes. However, due to the characteristics of the CTSP (namely total consistency and the fact that idle times/positions are not allowed), the way these permutations are used to generate solutions differs from what can be found in the literature. The constructive heuristic developed for the CTSP (Algorithm 1) iteratively tries to construct a feasible solution, repeating two phases—the first to define which positions are empty in each route, the second to assign positions to the nodes—until at least one feasible solution is generated.

In the first phase, the distribution of the 0 values in the solution matrix (i.e., the empty positions on the routes) is encoded by an integer value, hereinafter referred to as *type*. For a given CTSP instance, there are $Ntype = \prod_{l=1}^{m}(T - N^l + 1)$ different types, where $N^l = \#V^l$ and $T = \max_{l=1,\ldots,m}\{N^l\}$. For simplicity, we consider that the types range between 0 and $Ntype - 1$, where 0 is the default type, in which there are 0 empty positions before the start of the routes, while in type $Ntype - 1$ all routes visit their last client node at position T (all empty positions occur before the start of the routes).

After the empty positions are distributed according to a type in the first phase, the second phase consists of inserting client nodes in the first position available for them in the solution matrix, following the order in a random permutation/template. Position k is not available for node i if at least one of two conditions is verified: (i) at least one route l, with $S_i^l = 1$ (i.e., i must be visited in

Algorithm 1. Constructive heuristic

```
Input: S, n, m, T, max, Ntype;
bestcost ← M;
while bestcost = M do
    Generate a sorted random permutation of the client nodes, perm;
    if Ntype ≤ max then
        TYPES ← {0, ..., Ntype − 1} ;
    else
        TYPES ← random sample of the types, with dimension max ;
    end
    for type ∈ TYPES do
        Initialize solution matrix msol. Entries that should stay empty according to type get value 0, the remaining entries get value −1;
        all_nodes ← true;
        for h = 1, ..., n − 1 do
            i ← perm_h;
            inserted ← false;
            for k = 1, ..., T − 1 do
                if msol(l, k) = −1, ∀l : S_i^l = 1 then
                    msol(l, k) ← i, ∀l : S_i^l = 1;
                    inserted ← true;
                    break;
                end
            end
            if inserted = false then
                all_nodes ← false;
                break;
            end
        end
        if all_nodes = true then
            Compute the cost of msol, current_cost;
        else
            current_cost ← M;
        end
        if current_cost < bestcost then
            bestcost ← current_cost;
            bestsol(l, k) ← msol(l, k), ∀k = 1, ..., T, l = 1, ..., m;
        end
    end
end
Output: bestsol, bestcost
```

l) is already occupied by another node, j, in position k; (ii) at least one route l, with $S_i^l = 1$ has $msol(l, k) = 0$, that is, position k must remain empty according to the type considered. Therefore, the order of the nodes in the permutation indicates the order in which the nodes are inserted into the solution. However, because consistent nodes occupy the same position in the routes they are part of and empty positions are distributed before the nodes are inserted, the precedence relations that are verified in the permutation may not be verified in the solution matrix. For example, if route 1 starts in position 1, route 2 starts in position 2 and the first node in the permutation is (consistent) node i, visited on both routes, then i is inserted in position 2, while position 1 in route 1 will be occupied by a free node that follows i in the permutation. That is to say, unlike the traditional template approaches, in the CTSP, nodes in a given route in the solution matrix do not necessarily come in the same order as they appear in the permutation/template.

A feasible solution is found if all client nodes are inserted in the solution matrix. However, it is not always possible to do so and therefore several types are explored for the same template. Although ideally all types should be explored, some test instances for the CTSP (namely instances with 10 routes) result in problems with several millions of types, and exploring them all would

(a) Permutation vector (b) Sorted permutation vector

Fig. 3. Solution 2 from Fig. 2 can be obtained from the permutation vector in (a). By rearranging the nodes so that consistent nodes come before free nodes and precedence relations between consistent nodes and between free nodes are maintained, we obtain the sorted permutation vector in (b). In this example, both the unsorted and the sorted permutation vector can be decoded into the same solution matrix. However, this is not always the case.

result in prohibitive computational times. Therefore, a threshold, *max*, represents the number of types that we are willing to explore for the same permutation. If $Ntype < max$, then all types are explored. Otherwise, a random sample of the types, with dimension *max*, is explored. If, for a given template, at least one feasible solution is generated, the best solution found is presented as an output. Otherwise, the algorithm restarts with a new random permutation. After fine-tuning, the number of types to explore for a given template was set as $max = 50$.

Finally, imposing a limit on the number of types we are willing to assess in each iteration may not be enough to guarantee the algorithm's efficiency. In fact, in more complex cases it was verified that the constructive heuristic was time consuming, regardless of the value of *max*. For small values, although few types were explored in each iteration, a very large number of iterations was needed to find a feasible solution. For large values, many types were explored in each iteration, and, still, several iterations were needed to generate a feasible solution. To address this issue, *sorted random permutations* are used. Unlike fully random permutations, in sorted permutations, both consistent and free nodes are in a random order, but free nodes come after all of the consistent nodes. The idea is to insert the consistent nodes first, since they should occupy the same position in more than one route, and use the (more flexible) free nodes to populate the positions that are not occupied by consistent nodes, thus increasing the probability of obtaining a feasible solution in few or even one try, even if few types are explored. Preliminary tests showed that the use of sorted permutation allows for a significant improvement of computational times, without negatively affecting the quality of the solutions obtained with the ILS algorithm overall (even resulting, in many cases, in better final solutions). For the sake of implementation, sorted permutations are obtained from fully random permutations by moving consistent nodes to the beginning of the permutation, while maintaining all precedences between consistent nodes and between free nodes, as illustrated in Fig. 3.

4.2. Local search

This section describes the improvement heuristic iterated in the main step of the ILS. For the CTSP, a variable neighborhood search algorithm is proposed considering four different neighborhoods. The first three neighborhoods are searched in the solution matrix space, with only one of them allowing the type to change. The fourth neighborhood uses the template representation to further explore different solution types.

The Pos-pos neighborhood: A solution s' belongs to the Pos-pos neighborhood of solution s, Pos-pos(s), if it is obtained from s by swapping the nodes that occupy position k with all the nodes that occupy a different position, h in all routes. A move in this neighborhood is feasible if neither k nor h is empty on any route, or if both positions are empty on the same routes.

In the Pos-pos neighborhood, all nodes in position k are moved to position h and vice versa. However, it is also worth exploring moves in which not all routes are changed. This is done in the next neighborhood.

The Node-pos neighborhood: A solution s' belongs to the Node-pos neighborhood of solution s, Node-pos(s), if it is obtained from s by moving node i from position k to another position h. Observe that if i is a consistent node, then all nodes that occupy position h on the routes where i requires service must be moved to k. This means that a move in the Node-pos neighborhood can consist of swapping two nodes (if they are visited exactly on the same routes) or a node with several other nodes. To move node i to position h, two conditions must be verified: (i) position h is not empty on any route that visits i and (ii) for every node j that occupies position h before move, if i and j are both visited on at least one route, j cannot be visited on any route that does not visit i. Moves that do not satisfy these conditions might result either in the loss of total consistency or in empty positions between positions that are occupied, leading, therefore, to nonfeasible solutions.

Neither the Node-pos nor the Pos-pos neighborhood explores changes in the distribution of empty positions in the routes. This situation does not arise when all routes have the same length. However, when this is not the case, different types should be explored in the local search. Therefore, we propose the Type neighborhood.

The Type neighborhood: A solution s' belongs to the Type neighborhood of solution s, Type(s), if it is obtained from s by moving node i, visited in route l, from position $ps(l)$ to position $pf(l)+1$ in all routes it is visited in, or from position $pf(l)$ to position $ps(l)-1$ in all routes it is visited in, where $ps(l)$ is the first position occupied in route l, and $pf(l)$ is the last position in route l occupied by a client node. If i is moved from position $ps(l)$ to position $pf(l)+1$, the move is feasible if all routes in which i is visited start at position $ps(l)$ and finish at position $pf(l)$, and if $pf(l) < T$ (i.e., there are empty positions after position $pf(l)$). If i is moved from position $pf(l)$ to position $ps(l)-1$, the move is feasible if all routes in which i is visited start on position $ps(l)$ and finish in position $pf(l)$, and if $ps(l) > 1$ (i.e., there are empty positions before position $ps(l)$). For the sake of clarity, Fig. 4 provides an example of a Pos-pos neighbor, a Node-pos neighbor and a Type neighbor for a given solution.

For a move in the Type neighborhood to be feasible, all routes that visit the node being moved must start in the same position, finish in the same position, and have at least one empty position before the beginning (or after the end) of the route, raising concern regarding the effectiveness of the neighborhood. Preliminary tests show that, for most of the test instances, on average one or more solutions in the Type neighborhood are accepted per iteration of the ILS algorithm (meaning that not only there is at least one feasible Type neighbor of the local optimum obtained after applying the Node-pos and Pos-pos neighborhoods, but also there is at least one Type neighbor with a smaller cost than that local optimum). Still, it can be argued that the three neighborhoods described above ensure that the solutions searched are always feasible, at the cost of the search becoming more rigid. To allow for a more flexible search, another neighborhood was considered, the Type-template neighborhood, to explore solutions with different types while maintaining as many precedence relations between nodes as possible. This neighborhood is too large to be completely explored, and so at most a predefined number of neighbors will be considered.

The Type-template neighborhood: A solution s' belongs to the Type-template neighborhood of solution s, Type-temp(s), if it can be decoded from the template obtained from sorting a template

$$\begin{bmatrix} 6 & 3 & 2 & 4 & 5 \\ 0 & 3 & 2 & 7 & 8 \\ 6 & 9 & 2 & 4 & 0 \end{bmatrix}$$

(a) Solution 2

$$\begin{bmatrix} 6 & 4 & 2 & 3 & 5 \\ 0 & 7 & 2 & 3 & 8 \\ 6 & 4 & 2 & 9 & 0 \end{bmatrix}$$

(b) Pos-pos neighbor

$$\begin{bmatrix} 6 & 5 & 2 & 4 & 3 \\ 0 & 8 & 2 & 7 & 3 \\ 6 & 9 & 2 & 4 & 0 \end{bmatrix}$$

(c) Node-pos neighbor

$$\begin{bmatrix} 6 & 3 & 2 & 4 & 5 \\ 8 & 3 & 2 & 7 & 0 \\ 6 & 9 & 2 & 4 & 0 \end{bmatrix}$$

(d) Type neighbor

Fig. 4. Starting from solution 2 from Fig. 2, it is possible to illustrate the three different neighborhoods. The Pos-pos neighbor in (b) is obtained from Solution 2 by swapping all nodes in position 2 with all nodes in position 4. A Node-pos neighbor can be obtained by swapping two nodes that occupy the same routes (4 and 6, for instance) or one node with several other nodes. The solution in (c) corresponds to this latter case, having been obtained from Solution 2 by swapping node 3 with nodes 5 and 8. The solution in (d) is the only feasible Type neighbor for Solution 2, having been obtained by moving node 8 from the end to the beginning of the second route. Node 5 cannot be moved because there are no empty positions before the start of route 1. For similar reasons, node 6 cannot be moved, since although there is an empty position at the end of route 3, route 1 already finishes at the last available position.

that represents s so that consistent nodes appear before the free ones and such that the type of s' is different from the type of s.

The permutation/template used to search this neighborhood is constructed from the current solution, which is a local optimum for the Node-pos, Pos-pos, and Type neighborhoods. The template is initialized with all its entries empty, and a grid search is applied to the solution matrix, following the order of the routes and the order of the positions (only moving to the next position after all routes have been checked at the current position). When a node not yet in the template is found in the solution matrix, it is inserted into the first empty entry of the template. This procedure is repeated until all nodes have been inserted. The resulting template verifies all precedence relationships in the solution matrix and is guaranteed to be decoded into the solution matrix from which it was constructed. To illustrate, the template in Fig. 3a, was obtained from the solution matrix in Fig. 2c. Similar to what is done in the constructive heuristic in Section 4.1, the template is sorted so that consistent nodes appear before free nodes, and then up to *max* types are considered to generate new solutions from the sorted template. If several feasible solutions are generated, the best one is chosen and, if it is better than the current solution, it will substitute it, allowing the local search to continue.

The variable neighborhood search procedure: Algorithm 2 summarizes the variable neighborhood search algorithm with the four neighborhoods described above. The four neighborhoods are explored in sequence—first the Node-pos, then Pos-pos, followed by Type and finally Type-template—and in each one, the neighbor selected to continue the search is found following the *best improvement* rule. Each of the first three neighborhoods is explored until it reaches a local optimum or the best solution is updated. If the best solution is updated, the algorithm returns to search the Node-pos neighborhood from this solution. If a local optimum is reached without updating the best solution, the current solution becomes the starting point of the next neighborhood. The

Algorithm 2. Variable neighborhood search algorithm

```
Input: msol, tcost, type, Ntype;
improved ← true;
while improved = true do
    while improved = true do
        while improved = true do
            while improved = true do
                improved ← false;
                Compute the cost of all of the feasible moves in the Node-pos neighborhood;
                if The cost of the best move is lower than the cost of the current solution then
                    Accept the change and update msol and tcost;
                    improved ← true;
                end
            end
            Compute the cost of all of the feasible moves in the Pos-pos neighborhood;
            if The cost of the best move is lower than the cost of the current solution then
                Accept the change and update msol and tcost;
                improved ← true;
            end
        end
        Compute the cost of all of the feasible moves in the Type neighborhood;
        if The cost of the best move is lower than the cost of the current solution then
            Accept the change and update msol and tcost;
            improved ← true;
        end
    end
    Obtain permutation perm perm from msol;
    if Ntype ≤ max then
        TYPES ← {0, ..., Ntype − 1} ;
    else
        TYPES ← random sample of the types, with dimension max ;
    end
    for t ∈ TYPES do
        Try to generate a solution new_sol from perm, considering type t;
        if new_sol is feasible then
            Compute the cost of new_sol, new_cost;
            if new_cost < tcost then
                Update msol, type and tcost;
                improved ← true;
            end
        end
    end
end
Output: msol, tcost, type;
```

Type-template neighborhood is explored when the algorithm finds a solution that is simultaneously a local optimum for the Node-pos, Pos-pos, and Type neighborhoods. If the Type-template neighborhood can produce a better solution than the local optimum, the local search proceeds from this new solution, starting again from the Node-pos neighborhood. Otherwise, the local search stops.

To assess the relevance of the Type-template neighborhood, a simpler version of the variable neighborhood search is proposed, similar to Algorithm 2, but stopping as soon as a local optimum is found for the Type neighborhood. To distinguish the two cases, let *matheuristic (template)* refer to the matheuristic that considers a variable neighborhood search with all four neighborhoods, and *matheuristic* refer to the matheuristic that uses the simpler variable neighborhood search, with only the Node-pos, Pos-pos, and Type neighborhoods.

4.3. Perturbation

A perturbation is a transformation applied to the last solution of an iteration of the ILS algorithm to obtain the first solution of the next iteration. For a better performance of the heuristic overall, perturbations should not be too deterministic or too random, and there should also be a balance about the extent to which the solution is changed. Too small or deterministic perturbations may result in cycles in the search. In contrast, perturbations that are too large or random would be equivalent to random restarts of the local search.

Two different perturbations were proposed for the matheuristic. Both perturbations consist of destroy and repair operators applied to the template vector. In other words, both operators consist of generating a template from the current solution, removing the p nodes from the template, and reinserting them in random order while the other $n - p - 1$ nodes remain unchanged. The difference between the two operators lies in the rule used to choose the nodes that are removed from the template. In the *random perturbation*, the p nodes are chosen randomly, and all nodes have the same probability of being chosen. In the *greedy perturbation*, the impact of each node in the solution is computed by summing the cost of all arcs entering or leaving the node. Observe that if the same arc is used in several routes, its cost is multiplied by the total number of times that the arc is used in the solution.

After the p nodes are removed and reinserted, the resulting template is sorted and up to *max* different types are examined to try and generate new solutions. If it is possible to generate one or several new solutions, then the best solution is selected to become the first solution of the next iteration. Otherwise, a random restart is needed, that is, the constructive heuristic described in Section 4.1 is applied until a feasible solution is generated, in order to start the next iteration.

To try and make significant but not too large changes in the solutions, parameter p was defined as the integer number closest to $n/4$, the number of nodes divided by 4.

Some preliminary tests regarding these two operators in the context of a nonhybrid ILS algorithm can be found in Ponte (2023). The results show that, should only one perturbation be used, the random perturbation is the one that achieves better results. However, these results can be improved by randomly choosing between the two perturbations in each iteration of the algorithm. The tests were carried out for $p_r \in \{0.4, 0.5, 0.6\}$, with p_r being the probability of choosing the random perturbation (i.e., the probability of choosing the greedy perturbation is $1 - p_r$). The best results were obtained for $p_r = 0.4$.

4.4. Acceptance criterion and free node optimization

The acceptance criterion refers to the solution to which the perturbation is applied in each iteration and can be used to regulate the diversification of the algorithm. For clarity, let s_i be the solution to which the perturbation was applied at the end of iteration i and s_{i+1} the final solution at iteration $i + 1$. In the extreme case of diversification, s_{i+1} is always accepted, that is, the perturbation is applied to s_{i+1}, regardless of its cost. In the (opposite) extreme case of intensification, s_{i+1} is accepted only if its cost is less than the cost of s_i. Otherwise, the perturbation is applied to s_i.

One of the conclusions of the preliminary tests found in Ponte (2023) is that the ILS algorithm performs better when the random perturbation is used (either alone or, with even better results,

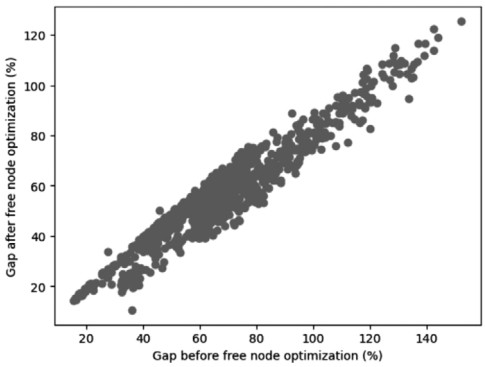


Fig. 5. Relationship between the gap of the input solution for free node optimization and the final solution after free node optimization.

when combined with the greedy perturbation). This suggests that the need to ensure feasibility results in a local search that does not explore large areas of the search space, thus benefiting from the introduction of randomness. Therefore, the acceptance criterion chosen for the ILS algorithm for the CTSP is the extreme diversification case. Consequently, it is necessary to keep a memory of the best solution found, which is the final output of the algorithm.

This issue also motivates the introduction of what will hereinafter be called *free node optimization*, the step that characterizes the algorithm under study as a matheuristic. The idea is to use an ILP formulation (in this case, the PQ model, as presented in Section 3) to solve an instance of rCTSP generated from the final solution of an iteration of the ILS algorithm. Since both the sequence of nodes and the distribution of empty positions are allowed to vary simultaneously, free node optimization quickly explores more than one neighborhood structure at the same time.

The optimal solution of an instance of the rCTSP is always feasible for the original CTSP problem, and, in many cases, it is significantly better than the input solution used to generate the rCTSP instance. However, applying free node optimization, even if using the most efficient formulation, to the final solution of every iteration of the ILS would result in prohibitive CPU times, due to the large number of iterations of the algorithm. Therefore, some further computational experiments were carried out to help decide in which situations free node optimization should be used.

Figure 5 provides a comparison between the gap of the input solution before free node optimization and the gap of the corresponding final solution after free node optimization. In this experiment a total of 984 pairs of solutions, distributed among the seven variants of test instances used on the computational experiment, were considered. The scatterplot in Fig. 5 shows a high correlation between the two gaps (in fact, the Pearson correlation coefficient between the two gaps is approximately 0.96), suggesting that worse initial solutions tend to result in worse final solutions. In other words, the quality of a solution seems to be more influenced by the distribution of consistent nodes, rather than by the distribution of free nodes.

Consequently, free node optimization should only be applied to promising input solutions. However, the question of how to identify promising solutions arises. Initially, the matheuristic was tested using free node optimization only when the best solution was updated. In other words, let s^* be the best solution and s' be the current solution at the end of an iteration. If the cost of s' is lower than

the cost of s^*, then free node optimization is applied using s' as input, to obtain solution s'', with a cost that is not greater than the cost of s'. The solution s'' becomes the best solution found so far, substituting s^*, as well as the current solution to which the perturbation is applied. Following this procedure resulted in an ILS algorithm that required short computational times but converged prematurely—because solutions obtained after free node optimization quickly become noncomparable with solutions obtained by local search, free node optimization was used very few times, and only in the initial iterations. To deal with this, the ILS algorithm was adapted to keep the memory of the best solution before free node optimization, s^*, and the best solution after free node optimization, s^{**}. If the current solution at the end of an iteration, s', has a lower cost than s^*, then s' substitutes s^* and is used as an input for free node optimization, resulting in the output solution s'', which becomes the new current solution and substitutes s^{**} if the cost of s^{**} is higher. Following this new procedure, the CPU time required by the algorithm increased slightly due to free node optimization being used more frequently, but the quality of the solutions provided by the heuristic also increased significantly. In both cases, it is important to note that solutions obtained through free node optimization usually cannot be further improved by local search, therefore, a perturbation is still applied before starting a new iteration.

4.5. Algorithm overview

Algorithm 3 provides an overview of the matheuristic. The initial solution is obtained through the constructive heuristic described in Section 4.1, that is, a template is randomly generated and decoded for at least a subset of the types, to generate one or several feasible solutions. If more than one solution is generated, the solution with the lowest cost, s, is selected. Local search is applied to s, as detailed in Section 4.2, resulting in final solution s^*. In the first iteration, s^* is saved as the best solution before optimization, *bsol*, and used as input for free node optimization. In subsequent iterations, free node optimization is used only when *bsol* is updated, that is, when the final solution of the local search, s^*, is better than *bsol*. The output solution of the free node optimization, s^{**}, is saved as the best solution after optimization, *opt_sol*, in the first iteration. In other iterations, *opt_sol* is updated only when free node optimization obtains a better solution. If the current iteration is not the last, a perturbation must be applied to the current solution. If free node optimization was applied, the current solution is s^{**}. Otherwise, the current solution is s^*. The choice between the two perturbations follows a Bernoulli distribution, where the random perturbation is selected with 40% probability. If the perturbation results in a feasible solution, it is used as the initial solution in the next iteration. Otherwise, the constructive heuristic must be used to obtain a new initial solution.

As mentioned before, to improve the algorithm's efficiency, sorted permutations (in the sense that consistent nodes appear before the free nodes) are used in every step that requires a template—the constructive heuristic, the Type-template neighborhood, and the perturbation. An attempt to further increase diversity was made by switching off template sorting for instances with just one solution type since, for those, two of the neighborhoods explored in the local search become irrelevant. For instances with one solution type and five routes, this allowed us to obtain better solutions without a significant increase of the computational times. For instances with one type and 10 routes, however, computational times doubled and solution quality did not increase. Due to these results,

Algorithm 3. The matheuristic

```
Input: Nit;
feasible ← false;
for it = 1, ..., Nit do
    if feasible = false then
        Use constructive heuristic to obtain solution s;
    end
    Use Local Search, starting from s, to obtain final solution s*, with cost tcost ;
    if it = 1 or tcost < bcost then
        bsol ← s*;
        bcost ← tcost;
        Apply free node optimization using s* as input, to obtain s**, with cost lcost;
        if it = 1 or lcost < opt_cost then
            opt_sol ← s**;
            opt_cost ← lcost;
        end
        s* ← s**;
    end
    if it < Nit then
        Randomly choose perturbation type, p;
        if p = random then
            Random perturbation to s*, to obtain s';
        end
        else
            Greedy perturbation to s*, to obtain s';
        end
        if s' is feasible then
            s ← s';
            feasible ← true;
        end
        else
            feasible ← false;
        end
    end
end
Output: opt_sol, opt_cost;
```

the matheuristic applies template sorting, except when solving instances with five routes and only one solution type. The number of types explored for the same template is $max = 50$.

To assess the impact of free node optimization on the performance of the matheuristic, an ILS algorithm that results from omitting the free node optimization step in Algorithm 3 was also tested. For completeness, and similar to what was proposed for the matheuristic in Section 4.2, computational results will also be shown for an ILS algorithm where the variable neighborhood search includes all four neighborhoods (*ILS (template)*) and an ILS algorithm where the Type-template neighborhood is omitted (*ILS*).

5. Computational experiment

To assess the performance of the ILP formulations for the CTSP, Gouveia et al. (2023) proposed a set of test instances ranging from 33 to 45 nodes, distributed among two, three, or five

Table 2
Generating 10-route test instances - gr96. To illustrate how to generate a 10-route test instance from a 5-route instance, consider cost matrix att48 (48 nodes) and variant v1. In this instance, originally, there are 12 consistent nodes, which are visited in all five routes, and each route has 7 free nodes that are unique for that route. The 10-route instance will have the double of the consistent nodes and the double of the free nodes. This is obtained by keeping 7 free nodes per route while considering a new total of 24 consistent nodes. Of the consistent nodes, six nodes are visited by the first five routes and six nodes are visited by the final five routes. To ensure that there is still interdependence between the routes (i.e., to avoid dividing the 10 routes into two independent groups of 5 routes each), six consistent nodes are visited by the first two and the final three routes, while the last six consistent nodes are visited by the remaining routes {3, 4, 5, 6, 7}.

Set	5 Routes	10 Routes
{1, 2, 3, 4, 5}	12 Nodes	6 Nodes
{6, 7, 8, 9, 10}	—	6 Nodes
{1, 2, 8, 9, 10}	—	6 Nodes
{3, 4, 5, 6, 7}	—	6 Nodes
Single routes	7 Free nodes/route	7 Free nodes/route

routes. In a first experiment, test instances used subjective costs, and the distribution of nodes across routes was customized to fit the application. A second, more generalized experiment was also performed using cost matrices adapted from the TSPLIB dataset (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html), which included seven variants to represent different node distributions: one variant (v7) corresponded to the original application, while the other six variants (v1–v6) represented varying levels of interdependence between routes. The framework of this second experiment was followed to assess the performance of the matheuristic, now considering larger instances, namely asymmetric matrices ftv47, ftv55, ftv64, and ftv70, as well as symmetric matrices att48, brazil58, st70, and eil76, for the tests with five routes. Some additional tests were carried out considering instances with 10 routes, again with 7 variants. These new instances are obtained from the ones with five routes in the following way:

1. Consider the number of (free) nodes that visit only route 1 in the instance with five routes, $n1$. In the instance with 10 routes, all routes will have $n1$ free nodes.
2. For each subset of the five original routes, S, let ns be the number of consistent nodes that are visited on all routes of S (and no routes outside of S). Consider also the set $S' = \{l + 5 : l \in S\}$. The instances with 10 routes have a total of $2ns$ consistent nodes. Of those nodes, $\frac{ns}{2}$ are visited by the routes in S and $\frac{ns}{2}$ are visited by the routes in S'. To distribute the remaining consistent nodes, sets S and S' are both divided in half, with $\frac{ns}{2}$ nodes being visited by the first half of S and the second half of S', and $\frac{ns}{2}$ nodes being visited by the second half of S and the first half of S'.
3. Adjustments are made (either by adding/removing up to two consistent nodes or by adding/removing up to two free nodes from each route) so that the total number of nodes fits the instances available in the TSPLIB.

Table 2 illustrates this procedure, for instance att48, variant v1. After Steps 1 and 2, we obtain a preliminary distribution of 95 nodes in total (94 client nodes plus the depot). By adding an extra free node to route 1 we obtain a total of 96 nodes, which fits instance gr96.

Therefore, the cost matrices considered for the experiment with the 10 routes were the asymmetric matrices kro124p and ftv170 and the symmetric matrices gr96, gr120, pr144, and u159. All the test instances can be accessed online (https://sites.google.com/view/constsp/home). The tests were carried out using the Concert Technology from CPLEX, version 12.9.0.0., in a computer with an Intel® Core™ i7-4790 CPU @ 3.60GHz processor, with 8GB of RAM.

Before presenting the results of the computational experiment, Section 5.1 provides some details on parameter tuning made to improve the efficiency of the free node optimization. The computational results are presented in Sections 5.2 and 5.3, with the former providing a more general comparison of the performances of the ILS and the matheuristic, with and without strong search, while the latter provides a more in-depth view of the performance of the best methodologies, by focusing on the test instances for which the optimal solution value (or a relatively good approximation) is known.

5.1. Parameter tuning

Despite the efficiency of the restricted PQ model, this methodology can be used several times throughout the ILS algorithm, with each run of the free node optimization taking between less than a second and several seconds, depending on the size of the instance and the quality of the initial solution. After preliminary testing, some fine-tuning was done to further reduce CPU times, namely: (i) Making the formulation as compact as possible. In particular, constraints (8)–(11) were not introduced as constraints in the model; instead, the variables that should have their value fixed at 0 are not generated. Variables referring to arcs in the form (i, i) are also not generated. (ii) The initial solution used to decide which position each consistent node must occupy is feasible for the rCTSP. Therefore, it is given to CPLEX as an input. (iii) The variable selection option was set to the *reduced costs* rule. (iv) Since the goal of using free node optimization is to quickly improve the local optima found in the ILS, the parameter absolute MIP gap tolerance was set to 0.01. Other CPLEX parameters kept their default values.

5.2. Computational results

This section provides a general comparison of four methodologies, namely the matheuristic as described in Algorithm 3, with and without the Type-template neighborhood (respectively, matheuristic (template) and matheuristic) and an ILS heuristic that results from omitting the free node optimization in Algorithm 3, again with and without the Type-template neighborhood (respectively, ILS (template) and ILS). The four methodologies are compared using three indicators—the computational time, measured in seconds, and two gaps, computed as follows:

$$lgap = \frac{\text{Heuristic value} - \text{Lower limit}}{\text{Lower limit}} \times 100, \quad ugap = \frac{\text{Heuristic value} - \text{Upper limit}}{\text{Upper limit}} \times 100$$

where *heuristic value* is the solution value provided by the heuristic method under assessment, *lower limit* is the optimal solution value, if known, otherwise it is given by the lower limit provided by the

Table 3
Computational times (seconds) and gaps concerning upper and lower limits (%)—averages per cost matrix

Matrix	ILS			ILS (template)			Matheuristic			Matheuristic (template)		
	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)
ftv47	12	4.84	1.44	16	4.80	1.40	13	4.06	0.69	18	4.03	0.66
ftv55	15	15.25	1.11	22	14.90	0.78	18	13.30	−0.60	24	13.49	−0.43
ftv64	20	20.79	−1.47	28	20.79	−1.47	26	17.44	−4.23	33	17.32	−4.31
ftv70	24	21.21	0.25	35	21.09	0.16	32	18.03	−2.37	42	17.90	−2.49
att48	16	11.38	−0.45	18	11.45	−0.39	16	11.28	−0.53	19	11.24	−0.57
brazil58	25	17.42	−2.79	29	17.44	−2.77	25	17.00	−3.12	29	17.09	−3.06
st70	38	32.41	−10.01	45	33.12	−9.49	41	31.09	−10.84	50	31.02	−10.88
eil76	45	31.67	−12.36	51	31.54	−12.43	52	29.68	−13.68	61	29.70	−13.64
kro124p	79	36.64	−6.70	89	36.67	−7.01	82	34.82	−7.74	97	34.77	−8.04
ftv170	197	48.92	−7.01	220	48.85	−7.02	300	35.72	−15.14	344	35.14	−15.24
gr96	62	22.29	−4.23	77	22.28	−4.24	65	21.67	−4.76	84	21.57	−4.82
gr120	123	44.10	−24.97	136	44.27	−25.34	130	40.53	−26.35	148	40.34	−26.51
pr144	193	59.45	−10.52	213	59.33	−10.59	271	53.30	−13.97	306	54.61	−13.25
u159	220	33.84	−8.39	240	32.90	−9.04	320	30.53	−10.66	352	30.03	−11.02
Average	76	28.59	−4.96	87	28.53	−5.04	99	25.60	−6.71	115	25.59	−6.71

APQ model at the end of 10 hours, and *upper limit* is the optimal solution value, if known (in which case $lgap = ugap$), otherwise given by the best feasible solution value found by the APQ model in 10 hours, if any solution was found within the time limit. Both gaps are measured in percentages, with *lgap* taking nonnegative values, whereas *ugap* can take negative values if the solution provided by the heuristic is better than the solution provided by model APQ after 10 hours.

The computational experiment included 14 cost matrices, 7 variants for each matrix, and repetitions for 5 different seeds, totaling 490 runs for each methodology under assessment. Therefore, and for clarity, results are represented through averages.

Table 3 shows the average results per cost matrix. Each line represents the average results across the 5 runs of each variant (totaling 35 runs). As expected, it is possible to see that the computational times increase both with the number of nodes and the number of routes. Also, Gouveia et al. (2023) reported that symmetric cost matrices generate instances that are more difficult to solve for the exact methods, and this also seems to be the case for the heuristic methods developed in this study—computational times are significantly larger for matrix att48 when compared to ftv47, matrix st70 when compared to ftv70, matrix gr120 when compared to kro124p, and even matrix u159 when compared to ftv170. The *lgap*s also tend to increase with the number of nodes, and the number of routes, as well as in instances with symmetric cost matrices, although with some exceptions—the average *lgap* for matrix eil76 is slightly smaller than for matrix st70, and the average *lgap* for matrix u159 is significantly smaller than for matrices pr144 and gr120. This behavior is not unexpected, since larger instances are more difficult to solve, and therefore the lower gaps provided by CPLEX tend to be worse. For similar reasons, smaller (more negative) values for *ugap*s in larger instances also reflect worse feasible solutions (upper limits) found by CPLEX in 10 hours. Still, these gaps allow us to compare the performance of the different methodologies. In particular, it can be seen that the introduction of the free node optimization allowed to improve, in many

Table 4
Computational times (seconds) and gaps concerning upper and lower limits (%)—averages per variant

Variant	ILS			ILS (template)			Matheuristic			Matheuristic (template)		
	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)
v1	83	25.40	−13.03	87	25.57	−12.87	98	23.51	−14.20	105	23.46	−14.22
v2	77	31.29	−5.54	93	31.24	−5.66	99	28.86	−6.90	119	28.54	−6.97
v3	77	32.24	−3.13	96	32.60	−3.04	100	29.24	−4.85	119	29.44	−4.72
v4	73	31.76	−2.23	89	31.44	−2.49	103	28.34	−4.09	125	28.47	−3.94
v5	79	31.26	−3.63	82	30.96	−3.90	112	26.95	−5.84	124	26.47	−6.13
v6	84	34.02	−6.32	91	33.97	−6.34	115	29.87	−8.99	132	30.39	−8.71
v7	62	14.13	0.29	71	13.94	0.11	69	12.45	−1.15	80	12.34	−1.25
Average	76	28.59	−4.96	87	28.53	−5.04	99	25.60	−6.71	115	25.59	−6.71

cases significantly, both the *lgap*s and the *ugap*s from the ILS to the matheuristic and from the ILS (template) to the matheuristic (template), albeit with a clear increase in computational times. On the other hand, the introduction of the Type-template neighborhood slightly increases computational times but does not seem to improve the quality of the solutions significantly. This is not surprising for the matheuristic, since the free node optimization includes exploring different types. Still, it also happens for the ILS algorithm, thus suggesting that this neighborhood does not add much to the type exploration already achieved in the constructive heuristic, perturbation, and Type neighborhood. Note that there are cost matrices for which the solution value found by a heuristic with the Type-template neighborhood is, on average, worse than the solution value found by the same methodology without this neighborhood, and this seems more recurrent for symmetric cost matrices (for the ILS, this fourth neighborhood decreases the quality of the final solution for $\frac{1}{6}$ of the asymmetric cost matrices and $\frac{1}{2}$ of the symmetric cost matrices, while for the matheuristic it decreases the quality of the final solution for $\frac{1}{6}$ of the asymmetric cost matrices and $\frac{3}{8}$ of the symmetric cost matrices). In most of the cost matrices, the heuristic methods outperform the APQ model in terms of quality of solutions (as shown by the negative average values for the *ugap*s). In particular, the matheuristic on average finds a better solution value (in several cases significantly better) than the APQ model in 10 hours in all cost matrices except for the smallest and easiest to solve matrix, ftv47.

Table 4 shows the average results per variant. Each line represents the average results across the 5 runs of each cost matrix (totaling 70 runs per method). To compare the performance of the heuristic methods on different variants, it is important to note that variant v1 is the one where the routes individually tend to visit more nodes and are more interdependent, while having few, or even just one solution type, variant v7 also has few or just one type, with the smallest and less interdependent routes across all variants, and variant v6 is the one where there are more solution types (Ponte, 2023, provides a more in-depth explanation of the variants, along with an analysis on the variation of the number of types across different variants). Accordingly, variant v7 and mostly variant v1 are the ones that require the smallest computational time, whereas variant v6 is mostly the one that requires the largest computational time. Variant v7 is also the one where the *lgap*s are smaller and the upper gaps are larger, which is justified by the fact that these instances tend to be

Table 5
Computational times (seconds) and gaps concerning upper and lower limits (%)—averages per seed

Seed	ILS			ILS (template)			Matheuristic			Matheuristic (template)		
	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)
1	76	28.66	−4.94	87	28.65	−4.97	102	25.80	−6.58	120	25.57	−6.75
2	77	28.71	−4.92	87	28.58	−5.08	99	25.89	−6.54	114	25.68	−6.70
3	76	28.63	−4.92	87	28.54	−5.01	97	25.67	−6.74	115	25.54	−6.72
4	76	28.53	−4.95	87	28.42	−5.10	98	25.43	−6.80	114	25.72	−6.60
5	76	28.41	−5.08	87	28.46	−5.03	101	25.24	−6.91	110	25.41	−6.76
Average	76	28.59	−4.96	87	28.53	−5.04	99	25.60	−6.71	115	25.59	−6.71

easier to solve for the exact methods, and thus these can provide better upper and lower bounds. On the other hand, variants v1 and v6 are the ones where the heuristic methods outperform model APQ more significantly, which is possibly due to the larger routes in the former and the largest number of types in the latter, both contributing to these instances being more challenging for exact methods.

Finally, Table 5 shows the average results per seed, with each line representing the average results across the 7 variants of each cost matrix (total of 98 runs per method). None of the methodologies shows significant fluctuations in the gaps, suggesting that the four methods are robust. The matheuristics (with and without the Type-template neighborhood) seem to have a slightly larger variation in the average computational times, when compared to the ILS algorithm, which can be explained by the fact that, depending on the quality of the initial solution, the time spent on one run of free node optimization can vary significantly, even for the same test instance.

Some additional insights obtained from the computational experiment include the following.

- The heuristic methods tend to update the best local optimum on average 8–12 times (this means that, on average, the ILS heuristic updates the best found solution 8–12 times and free node optimization is used the same number of times in the matheuristic). This behavior is quite robust for the five tested seeds and does not change from variant to variant.
- The last time the best local optimum is updated (with resulting free node optimization, if using the matheuristic) tends to occur, on average, between iterations 8000 and 14,000, with the matheuristic usually converging on earlier iterations than the ILS, although not significantly. This suggests that 25,000 iterations are sufficient for the algorithm to converge (and possibly could be reduced to decrease computational times without affecting the quality of the solutions). Unsurprisingly, the variants with more solution types (v4, v5, and v6) are the ones where the methodologies tend to converge in later iterations. Again, the number of iterations until convergence does not vary much from seed to seed.
- Regarding the matheuristic, the best solution value (after free node optimization) is found, on average, between iterations 5000 and 10,000, suggesting that in several cases the final solution is found in a significantly earlier iteration than the last free node optimization. More precisely, the final solution of the matheuristic is obtained in the last free node optimization in about 55% of all runs with the Type-template neighborhood and 58% of all runs without this neighborhood. In the remaining cases, the differences between the iteration in which the final solution was found and the iteration in which the last free node optimization was performed are usually significant.

Table 6
Computational times and gaps for matheuristic and matheuristic (template)—in-depth analysis

Instance	Matheuristic			Matheuristic (template)		
	Time (seconds)	*lgap* (%)	*ugap* (%)	Time (seconds)	*lgap* (%)	*ugap* (%)
ftv47v1	14	0.00	0.00	19	0.00	0.00
ftv47v2	14	0.48	0.48	18	0.22	0.22
ftv47v3	14	4.45	−0.06	19	4.40	−0.52
ftv47v4	13	3.01	3.01	19	3.23	3.23
ftv47v7	12	1.38	1.38	15	1.24	1.24
ftv55v7	15	0.59	0.59	20	1.18	1.18
att48v2	17	6.51	0.00	20	6.51	0.00
att48v7	14	0.06	0.05	17	0.10	0.09
gr96v7	57	0.01	0.00	72	0.01	0.00

This behavior does not contradict Fig. 5, since it generally arises from one of the following two situations: (i) The solution obtained from the last free node optimization is not the final solution. In this case, the solution after the last free node optimization is usually not significantly worse than the final solution, and the latter tends to be found in one of the last runs of the free node optimization. However, as the algorithm progresses and updates the best solution, it becomes increasingly more difficult to find even better solutions, thus the number of iterations between the final runs of free node optimization can be quite large. (ii) The solution obtained from the last free node optimization is the final solution, but the same solution had already been obtained from an earlier run of free node optimization, from a different input solution.

The results show that free node optimization has a significant positive impact on the performance of the matheuristic, although at the cost of an increase in computational times, whereas the Type-template neighborhood does not seem to have a significant impact on the performance of the methodology. Section 5.3 provides a more in-depth analysis of the matheuristic and the matheuristic (template), focusing on test instances for which the optimal solution value (or a relatively good approximation) is known.

5.3. In-depth analysis

Table 6 shows more detailed results for the matheuristic, focusing on the instances for which the APQ model was able to find the optimal solution value (or a good approximation) in 10 hours, namely most of the variants of matrix ftv47, variant v7 of matrices ftv55, att48, and gr96, and variant v2 of matrix att48. In each line, results are presented as averages across the five runs of an instance (one for each seed). The instances for which the optimal solution value is known are the ones where *lgap* equals *ugap*, that is, ftv47v1, ftv47v2, ftv47v4, ftv47v7, and ftv55v7. For these instances, the matheuristic was able to find good solutions, even consistently obtaining the optimal solution for ftv47v1, in every seed. The optimal solution for instance ftv47v3 is not known yet, but the lower limit provided by CPLEX allows us to conclude that the gap between the heuristic value and the optimal value is at most 4.5%, with the heuristic obtaining slightly better solutions than the

APQ model at the end of 10 hours. Similarly, for instance att48v2, it is possible to conclude that the gap between the heuristic value and the optimal value is at most 6, 51%, with the matheuristic consistently obtaining the same solution as model APQ, in every seed. Finally, the solutions obtained by the heuristic for instances att48v7 and gr96v7 are very close to optimality, if not optimal.

6. Conclusions

This study focused on the development of a matheuristic for the TSP with positional consistency constraints. To this effect, an ILS-based matheuristic was developed and implemented, which includes a constructive heuristic that tries to generate solutions with different types from a random permutation of the nodes and, if able to generate several feasible solutions, returns the one with the smallest cost; a variable neighborhood search algorithm with four different neighborhoods: three of them making changes in the solution matrix space, the fourth one using a template representation to explore solutions with different types; and a formulation for a restricted version of the CTSP, where the positions for consistent nodes are known and fixed, while free nodes and empty positions are allowed to vary. Preliminary tests showed that applying this formulation using a local optimum from the variable neighborhood search as input allowed to obtain output solutions with significantly better quality and that the linear correlation between the quality of the input solution and the quality of the output solution is very strong, which motivated the decision of applying the model only when the best local optimum is updated, thus avoiding prohibitive computational times.

A computational experiment was carried out to assess the performance of the matheuristic, considering a set of test instances, ranging from 48 to 171 nodes, with 5 or 10 routes. The matheuristic was shown to perform well in these tests, being able to find, on average, in around 100 seconds better solutions (in many cases considerably better) than the ones obtained by the APQ model proposed by Gouveia et al. (2023), in 10 hours. The free node optimization step (using a local optimum as input for the model for the rCTSP) was shown to positively impact, also significantly, the quality of the final solution for the heuristic (when compared to an ILS algorithm that uses the same constructive heuristic, the same variable neighborhood search algorithm and the same perturbation as the matheuristic), whereas the additional type-template neighborhood was shown to not affect significantly the performance of the algorithms under assessment. Finally, for the subset of instances for which the optimal solution is known, the matheuristic was able to obtain optimal or near-optimal solutions in most cases.

Future lines of research may include the study of other forms of synchronization that are relevant for healthcare services applications, namely, evaluating whether current methodologies for the CTSP can be adapted to address such forms of synchronization and, if so, how they affect the performance of the methodologies. Some possibilities include limiting the number of routes that can assign the same node (or "similar" nodes) to the same position, to model space or equipment scarcity; idle times/positions; precedence relationships (for instance, a doctor can only examine a patient after he has been seen by the nurse), consistency constraints where the distribution of the nodes among the routes is not (completely) known, to model patients that have not been affected to a specific family practitioner; or even cases where a node/set of nodes must be covered by at least one route in every position to model, for instance, situations where at least one healthcare professional must be available to serve walk-ins.

Acknowledgments

This research was supported by Portuguese National Funding from FCT – Fundação para a Ciência e a Tecnologia under project UID/04561/2025 and grant SFRH/BD/146812/2019.

Open access publication funding provided by FCT (b-on).

References

Anderluh, A., Hemmelmayr, V.C., Nolz, P.C., 2017. Synchronizing vans and cargo bikes in a city distribution network. *Central European Journal of Operations Research* 25, 2, 345–376.

Andersson, H., Duesund, J.M., Fagerholt, K., 2011. Ship routing and scheduling with cargo coupling and synchronization constraints. *Computers & Industrial Engineering* 61, 4, 1107–1116.

Braekers, K., Hartl, R.F., Parragh, S.N., Tricoire, F., 2016. A bi-objective home care scheduling problem: analyzing the trade-off between costs and client inconvenience. *European Journal of Operational Research* 248, 2, 428–443.

Bredström, D., Rönnqvist, M., 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research* 191, 1, 19–31.

Cappanera, P., Requejo, C., Scutellà, M.G., 2020. Temporal constraints and device management for the skill VRP: mathematical model and lower bounding techniques. *Computers & Operations Research* 124, 105054.

Ceschia, S., Di Gaspero, L., Rosati, R.M., Schaerf, A., 2026. Multi-neighborhood simulated annealing for the home healthcare routing and scheduling problem. *International Transactions in Operational Research* 33, 1, 38–67.

Decerle, J., Grunder, O., El Hassani, A.H., Barakat, O., 2017. A general model for the home health care routing and scheduling problem with route balancing. *IFAC-PapersOnLine* 50, 1, 14662–14667.

Drexl, M., 2012. Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science* 46, 3, 297–316.

Emadikhiav, M., Bergman, D., Day, R., 2020. Consistent routing and scheduling with simultaneous pickups and deliveries. *Production and Operations Management* 29, 8, 1937–1955.

En-nahli, L., Afifi, S., Allaoui, H., Nouaouri, I., 2016. Local search analysis for a vehicle routing problem with synchronization and time windows constraints in home health care services. *IFAC-PapersOnLine* 49, 12, 1210–1215.

Feillet, D., Garaix, T., Lehuédé, F., Péton, O., Quadri, D., 2014. A new consistent vehicle routing problem for the transportation of people with disabilities. *Networks* 63, 3, 211–224.

Fink, M., Desaulniers, G., Frey, M., Kiermaier, F., Kolisch, R., Soumis, F., 2019. Column generation for vehicle routing problems with multiple synchronization constraints. *European Journal of Operational Research* 272, 2, 699–711.

Frifita, S., Masmoudi, M., 2020. VNS methods for home care routing and scheduling problem with temporal dependencies, and multiple structures and specialties. *International Transactions in Operational Research* 27, 1, 291–313.

Goeke, D., Roberti, R., Schneider, M., 2019. Exact and heuristic solution of the consistent vehicle-routing problem. *Transportation Science* 53, 4, 1023–1042.

Gouveia, L., Paias, A., Ponte, M., 2023. The travelling salesman problem with positional consistency constraints: an application to healthcare services. *European Journal of Operational Research* 308, 3, 960–989.

Groër, C., Golden, B., Wasil, E., 2009. The consistent vehicle routing problem. *Manufacturing & service operations management* 11, 4, 630–643.

Ioachim, I., Desrosiers, J., Soumis, F., Bélanger, N., 1999. Fleet assignment and routing with schedule synchronization constraints. *European Journal of Operational Research* 119, 1, 75–90.

Kovacs, A.A., Golden, B.L., Hartl, R.F., Parragh, S.N., 2014a. Vehicle routing problems in which consistency considerations are important: a survey. *Networks* 64, 3, 192–213.

Kovacs, A.A., Golden, B.L., Hartl, R.F., Parragh, S.N., 2015a. The generalized consistent vehicle routing problem. *Transportation Science* 49, 4, 796–816.

Kovacs, A.A., Parragh, S.N., Hartl, R.F., 2014b. A template-based adaptive large neighborhood search for the consistent vehicle routing problem. *Networks* 63, 1, 60–81.

Kovacs, A.A., Parragh, S.N., Hartl, R.F., 2015b. The multi-objective generalized consistent vehicle routing problem. *European Journal of Operational Research* 247, 2, 441–458.

Kummer, A.F., de Araújo, O.C., Buriol, L.S., Resende, M.G., 2024. A biased random-key genetic algorithm for the home health care problem. *International Transactions in Operational Research* 31, 3, 1859–1889.

Lespay, H., Suchan, K., 2021. A case study of consistent vehicle routing problem with time windows. *International Transactions in Operational Research* 28, 3, 1135–1163.

Lian, K., Milburn, A.B., Rardin, R.L., 2016. An improved multi-directional local search algorithm for the multi-objective consistent vehicle routing problem. *IIE Transactions* 48, 10, 975–992.

Liu, W., Dridi, M., Fei, H., El Hassani, A.H., 2021. Hybrid metaheuristics for solving a home health care routing and scheduling problem with time windows, synchronized visits and lunch breaks. *Expert Systems with Applications* 183, 115307.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search. In *Handbook of Metaheuristics*. Springer, Berlin, pp. 320–353.

Luo, Z., Qin, H., Che, C., Lim, A., 2015. On service consistency in multi-period vehicle routing. *European Journal of Operational Research* 243, 3, 731–744.

Mancini, S., Gansterer, M., Hartl, R.F., 2021. The collaborative consistent vehicle routing problem with workload balance. *European Journal of Operational Research* 293, 3, 955–965.

Mankowska, D.S., Meisel, F., Bierwirth, C., 2014. The home health care routing and scheduling problem with interdependent services. *Health Care Management Science* 17, 1, 15–30.

Nolz, P.C., Absi, N., Feillet, D., Seragiotto, C., 2022. The consistent electric-vehicle routing problem with backhauls and charging management. *European Journal of Operational Research* 302, 2, 700–716.

Parragh, S.N., Doerner, K.F., 2018. Solving routing problems with pairwise synchronization constraints. *Central European Journal of Operations Research* 26, 443–464.

Picard, J.C., Queyranne, M., 1978. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research* 26, 1, 86–110.

Ponte, M., 2023. The travelling salesman problem with positional consistency constraints. Ph.D. thesis, Universidade de Lisboa, Faculdade de Ciências.

Stavropoulou, F., Repoussis, P.P., Tarantilis, C.D., 2019. The vehicle routing problem with profits and consistency constraints. *European Journal of Operational Research* 274, 1, 340–356.

Subramanyam, A., Gounaris, C.E., 2016. A branch-and-cut framework for the consistent traveling salesman problem. *European Journal of Operational Research* 248, 2, 384–395.

Subramanyam, A., Gounaris, C.E., 2018. A decomposition algorithm for the consistent traveling salesman problem with vehicle idling. *Transportation Science* 52, 2, 386–401.

Tarantilis, C.D., Stavropoulou, F., Repoussis, P.P., 2012. A template-based tabu search algorithm for the consistent vehicle routing problem. *Expert Systems with Applications* 39, 4, 4233–4239.

Tellez, O., Vercraene, S., Lehuédé, F., Péton, O., Monteiro, T., 2022. The time-consistent dial-a-ride problem. *Networks* 79, 4, 452–478.

Wang, K., Zhen, L., Xia, J., Baldacci, R., Wang, S., 2022. Routing optimization with generalized consistency requirements. *Transportation Science* 56, 1, 223–244.

Xu, Z., Cai, Y., 2018. Variable neighborhood search for consistent vehicle routing problem. *Expert Systems with Applications* 113, 66–76.