Mathematics

https://www.aimspress.com/journal/Math

AIMS Mathematics, 11(1): 2458–2480.
DOI: 10.3934/math.2026100
Received: 18 September 2025
Revised: 29 December 2025
Accepted: 16 January 2026
Published: 26 January 2026

Research article

An agile optimization algorithm for the tourist trip design problem with type-covering constraints

Xabier A. Martin[1], **Javier Panadero**[2] **and Angel A. Juan**[1,*]

[1] CIGIP - ValgrAI, Universitat Politècnica de València, Alcoy, Spain

[2] Dept. of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Bellaterra, Spain

* **Correspondence:** Email: ajuanp@upv.es.

Abstract: The tourist trip design problem with type-covering constraints (TTDP-TC) is a novel variant of the well-established orienteering problem (OP) designed to address the complex preferences of tourists planning multi-day trips. Unlike classical routing problems, which require visiting all points of interest (POIs), the TTDP-TC allows selective visitation based on perceived value, subject to a maximum travel time constraint. This variant introduces a type-covering requirement, ensuring that each trip includes at least one POI of every specified type, adding a layer of complexity to the optimization process. In this paper, an agile optimization algorithm to solve the TTDP-TC efficiently is proposed, which aims to maximize the total profit collected from visited POIs while ensuring compliance with type-covering requirements and travel time limits. Our approach applies the coverage rules to the overall routing plan, instead of just to singular itineraries as in other recent studies. The algorithm's performance is validated through extensive computational experiments, demonstrating its ability to generate high-quality solutions within short computational times. To further validate the competitiveness of our approach, an exact method to compare the results of our approach is also implemented. The proposed approach showcases significant potential for practical applications in tourist decision support systems, offering a flexible and robust solution for planning enriched and diverse tourist experiences.

Keywords: tourism planning; orienteering problem; heuristics; biased-randomization; agile optimization
Mathematics Subject Classification: 90C27, 90C59

1. Introduction

Whether it be at the local or global scale, involving individuals or whole organizations, the acts of traveling and transporting are crucial aspects of human life. Accordingly, the field of combinatorial optimization describes a wide variety of routing problems that represent the extensive, complex logistic choices movement entails. In classical routing problems, all nodes or points of interest (POIs) must be visited regardless of their importance or potential benefit to the decision-maker. In contrast, routing problems with profit model scenarios allow the decision-maker to selectively choose which POIs to visit based on their perceived value or benefit, subject to budget constraints. Apart from the interest in finding the solution to distribution problems, the significance of the prize-collecting routing is also related to its applicability in addressing tourist trip design problems (TTDPs) [1]. Indeed, many different preferences or demands entailed by the design of good touristic trips can be modeled through specific variants of routing problems with profits. These modeling choices can extend to address multiple-day trips. One of the features that can be discussed in the study of TTDPs refers to the characterization of the POIs, i.e., the addition of a label to the nodes of the network to specify whether a given node represents a museum, a religious site, a food attraction, a natural reserve, and so forth. This additional information indirectly generates a partition of the set of POIs in clusters that collect all the nodes of a given type. In tourist decision support systems, the use of this information is discussed to either limit the maximum number of different types that can be visited, in a single day or the whole trip, or to require the visit of at least a POI of a mandatory type [2].

This paper focuses on a variant of the TTDP, namely the TTDP with type-covering constraints (TTDP-TC), a routing problem in which the objective is not only to maximize the total profit collected from visiting selected POIs, but also to ensure that the overall itinerary includes at least one POI of every specified type. Under an n-day planning horizon, a solution to the TTDP-TC consists of a set of n routes whose lengths cannot exceed a given time limit, and such that at least one POI of each type appears in the set of routes. Unlike related formulations that enforce coverage constraints at the level of individual itineraries, the TTDP-TC integrates type coverage across the entire routing plan, allowing a balanced distribution of POI types over multiple days or routes. This modeling choice is particularly relevant in multi-day itinerary planning, as it allows each daily route to be planned flexibly while ensuring coverage across the entire itinerary, improving variety and overall tourist experience. Moreover, we adopt a hard type-covering constraint as hard coverage has direct practical relevance in settings where coverage is mandatory such as travel-agency packages with contractual inclusions, curated cultural or educational programs that must visit certain categories, or conference organized event itineraries that guarantee exposure to predefined types of venues. Figure 1 depicts an illustrative example of a solution to a TTDP-TC instance. In this scenario, the origin and destination depots are represented by red squares, while POIs are shown as colored nodes (and icons) indicating their visit type. The example considers a two-day plan, represented by two routes corresponding to the paths taken by tourists to visit the selected POIs. To solve this problem, we propose an agile biased-randomized iterated local search (BR-ILS) approach, designed to produce high-quality solutions rapidly so that the planning system can promptly respond to tourist demands.

The main contributions of this paper are given next: *(i)* a novel variant of the TTDP, namely the TTDP with type-covering constraints, is proposed; *(ii)* a mathematical model describing the details of the aforementioned version is provided; and *(iii)* an 'agile' optimization algorithm, combining a

BR-ILS with different local search operators, is proposed to solve the problem.

The remainder of the paper is structured as follows. Section 2 offers a review of similar works in the literature. Section 3 formally describes the TTDP-TC and discusses the advantages of the formulation. Section 4 presents the proposed solution approach, which is designed to provide high-quality results in short computational times. Section 5 discusses a series of computational experiments carried out to test the effectiveness of our methodology. Section 6 analyzes in detail the achieved results. Lastly, Section 7 sketches the conclusive remarks of this work and proposes some future lines of research.

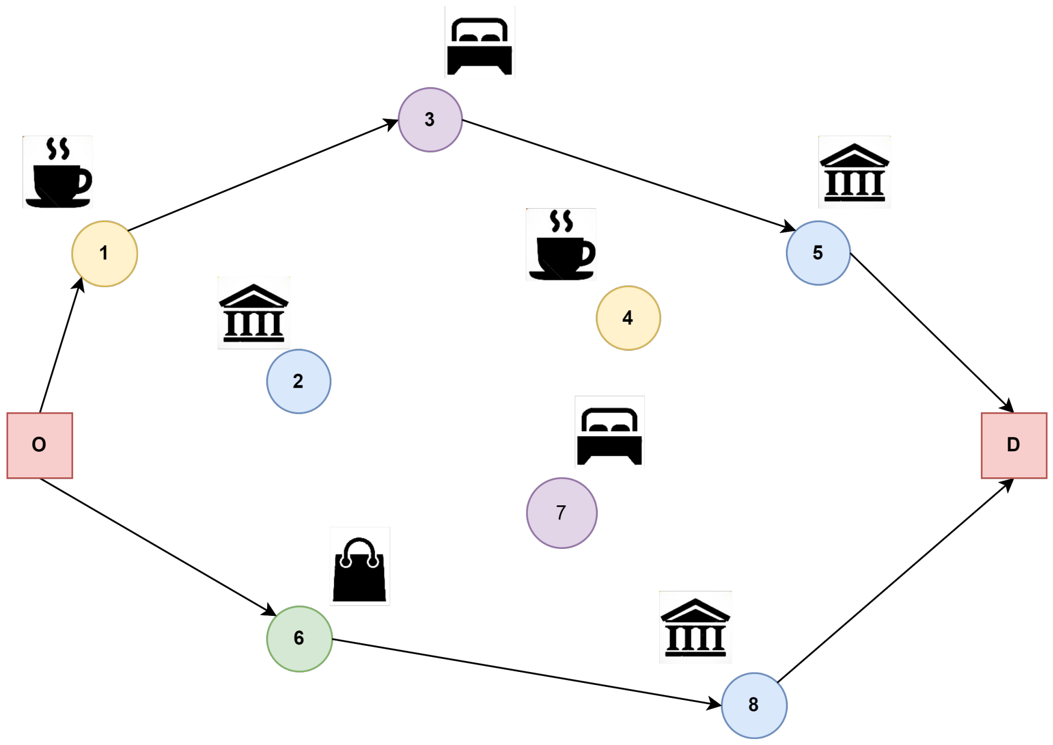

Figure 1. Schema of a possible solution to a TTDP-TC instance.

2. Related work

One of the most prominent examples of routing problems with profits is the orienteering problem (OP) [3], which draws inspiration from the orienteering game, in which participants are given a map and a list of checkpoints and must plan a route that visits a subset of the checkpoints to maximize their score within a given time limit. Whenever the optimal decision involves more than a single individual or vehicle, the OP generalizes to the team orienteering problem (TOP). Several research efforts have addressed OP or TOP variants including features such as time windows [4], mandatory visits [5], profits that depend on either the service time [6] or the time of arrival at a node [7], path-dependent travel times [8], or soft driving-range limitations [9]. Further studies have also included stochastic versions of the TOP, which aimed to model the intrinsic uncertainties that affect either the travel times [10–12], or the benefits gained by visiting the nodes [13]. See the works of [14, 15] for thorough surveys on the OP and its variants.

Notably, recent research has focused on versions of the OP in which the nodes are grouped into clusters. This modeling choice represents a flexible approach to include additional information concerning either geographical proximity or some other form of close relationships between the nodes. The set orienteering problem (SOP), originally defined by [16], describes a scenario in which the nodes to be served are partitioned into mutually exclusive clusters, and the profit, associated with clusters, is collected if at least a node of the cluster is visited. In the scientific literature, the SOP has been mainly tackled through a variety of metaheuristic algorithms, such as a variant of tabu search [16], biased random key genetic algorithm [17], and variable neighborhood search [18]. Notably, [19] proposed a multi-start approach that matches or improves state-of-the-art algorithms by combining the principles of tabu search and the solution of suitably defined integer programming models. Moreover, extensions of the SOP have been developed to accommodate scenarios involving multiple vehicles. The set team orienteering problem (STOP) was addressed by [20] with a branch-and-price method and a large neighborhood search algorithm, demonstrating significant improvements in solving large-scale instances of the problem. Recently, the STOP was extended to handle time windows and solved using an adaptive large neighborhood search algorithm and compared to results from the commercial solver CPLEX [21]. The main application discussed in the SOP literature relates to first-mile delivery, in which a carrier supplies all the goods required in a supply chain, modeled as a cluster, to a single node, who subsequently will take the burden of distributing the products to other retailers.

Differently, the clustered orienteering problem (COP), as introduced by [22], addresses scenarios where service contracts between a provider and the nodes within a supply chain mandate that the entire cluster of nodes be serviced for the contract to be valid. Accordingly, the profits, once again related to the clusters, are collected once all the nodes of a cluster are visited. In this case, the scientific literature also presents examples of exact approaches and metaheuristic algorithms to address the problem, such as branch-and-cut [22], a hybrid heuristic [23], and a hybrid evolutionary algorithm [24]. Notably, [24] proposed an evolutionary algorithm that integrated a backbone-based crossover operator and a destroy-and-repair mutation operator for search diversification and a solution-based tabu search procedure reinforced by a reinforcement learning mechanism for search intensification. Similarly, the COP was extended to handle scenarios involving multiple vehicles. The clustered team orienteering problem (CTOP) has been addressed by [25] with a cutting planes algorithm and a hybrid heuristic that combines an adaptive large neighborhood search and an effective split procedure. Practical applications of the COP are particularly relevant in supply chain logistics, where service contracts often stipulate that all members of a cluster must be served to fulfill contractual obligations.

Lastly, [26] consider a variant of the generalized traveling salesman problem (GTSP), i.e., the TSP that presents the nodes partitioned in clusters, in which the salesperson can choose the nodes to serve in order to collect their associated profits. In this problem, referred to as selective GTSP (SGTSP), neither all the nodes nor all the clusters must be visited in the tour, yet at most a single node of each cluster can be served. In their work, the authors formally introduce the problem and eight different mathematical formulations, showing that the best-performing ones were the sequence and time-based formulations, with auxiliary flow variables defined between nodes. To the best of our knowledge, Table 1 summarizes the current taxonomy of OP variants with clusters through their relevant features. The inclusion of constraints to visit nodes of different clusters represents a novel aspect in comparison to previous studies. As far as we are aware, no other work has included these type-covering constraints in routing problems.

Table 1. Summary of the OP variants with clusters.

Variant	Disjoint Clusters	Profit		Vehicles
		Associated to	Collected if	
SOP	Yes	Clusters	At least a node is visited	1
STOP	Yes	Clusters	At least a node is visited	> 1
COP	No	Clusters	All the nodes are visited	1
CTOP	No	Clusters	All the nodes are visited	> 1
SGTSP	Yes	Nodes	Each cluster is visited at most once	1

In recent years, several approaches have been proposed with similar coverage constraints. In particular, [27, 28] examined these restrictions within the OP with time windows and service times, where minimum and maximum coverage requirements are imposed on each itinerary. Specifically, nodes are associated with a category, and the final itinerary is required to contain at least a minimum and at most a maximum of specific categories. Similarly, [29, 30] addressed the same coverage constraints for the variant with a team of vehicles and though their formulations applied the coverage rules to singular routes rather than to the overall routing itinerary. The proposed TTDP-TC focused on planning routes for a single tourist, and enforced type coverage across the entire set of planned routes. In single-day planning, this behaves similarly to clustered TTDP formulations, where coverage is effectively per-route. However, in multi-day planning, individual routes are not required to be type-complete, allowing each daily route to be flexibly optimized according to geographic proximity and tourist preferences without forcing low-value or distant POIs into a route simply to satisfy per-day coverage. This distinction has several practical implications: it improves the overall distribution of POI types across the trip, enhances variety and overall tourist experience, reduces unnecessary travel, and allows for more robust and adaptable itineraries in the case of changes or disruptions. Overall, this approach produces more realistic, efficient, and user-friendly multi-day itineraries compared to per-route coverage models.

3. Problem formulation

The TTDP-TC is presented as an extension of the model proposed by [31], which is formulated on a directed graph $G = (V, A)$ where V represents the set of vertices and A represents the set of directed arcs. The set of vertices $V = \{0, 1, \dots, n, n+1\}$, where 0 and $n+1$ are the origin and destination depots, respectively, and the set $N = \{1, 2, \dots, n-1, n\}$, indicating the n POIs. The set of arcs $A = \{(i, j) \in A, i \in V, j \in V, i \neq j\}$ corresponds to the set of ordered pairs of vertices. For each arc $(i, j) \in A$, a non-negative travel time $t_{ij} > 0$ is defined, which is assumed to satisfy the triangular inequality. Each POI $i \in N$ has an associated profit $p_i > 0$, which is collected the first time the POI is visited. Moreover, m homogeneous vehicles start at the origin depot, visit some POIs, and end at the destination depot before a maximum allowed travel time $T_{\max}$.

In addition to the basic constraints, we define a set of K clusters $C = \{C_1, C_2, \dots, C_k\}$, where each $C_i \subseteq N$ represents a subset of vertices. The set C forms a cover of N, where $\cup_{i=1}^{K} C_i = N$. Notice that, it is possible to have POIs who belong to several clusters at the same time (e.g., a historic cathedral that functions both as a religious site and a museum, or a vineyard that can be classified as both a food attraction and an historical site). Therefore, a single visit can legitimately satisfy multiple cluster requirements when appropiate.

Moreover, any vehicle can visit POIs from many clusters, and multiple vehicles can visit POIs in a single cluster. There is no restriction on the order in which the visits must occur; that is, a vehicle can alternate visits to POIs from various clusters. The requirement is that each set C_i must have at least one of its POIs visited by at least one vehicle.

Before proceeding further, we introduce the following decision variables to model the TTDP-TC. Let x_{ij} be a binary variable that equals 1 if a vehicle travels through arc (i, j), and 0 otherwise. Let y_i be a binary variable that equals 1 if POI i is visited, and 0 otherwise. Let z_{ij} be a continuous variable that represents the arrival time at vertex j of a vehicle coming from vertex i. For each set $S \subseteq N$, let $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ and $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$ be the set of arcs leaving and entering the set S, respectively, with $\delta^+(i) = \delta^+(\{i\})$ and $\delta^-(i) = \delta^-(\{i\})$. Thus, we present the mathematical formulation for the TTDP-TC as follows:

$$\max \sum_{i \in N} p_i y_i \tag{3.1}$$

subject to

$$\sum_{j \in N} x_{0j} = \sum_{i \in N} x_{i,n+1} = m \tag{3.2}$$

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} = \sum_{(i,j)\in\delta^+(i)} x_{ij} = y_i \qquad \forall i \in N \tag{3.3}$$

$$z_{0j} = t_{0j} x_{0j} \qquad \forall j \in N \tag{3.4}$$

$$\sum_{(i,j)\in\delta^+(i)} z_{ij} - \sum_{(j,i)\in\delta^-(i)} z_{ji} = \sum_{(i,j)\in\delta^+(i)} t_{ij} x_{ij} \qquad \forall i \in N \tag{3.5}$$

$$z_{ij} \geq (t_{0i} + t_{ij}) x_{ij} \qquad \forall (i, j) \in A \setminus \{(0, n+1)\} \tag{3.6}$$

$$z_{ij} \leq (T_{\max} - t_{j,n+1}) x_{ij} \qquad \forall (i, j) \in A \setminus \{(0, n+1)\} \tag{3.7}$$

$$\sum_{i \in C_i} y_i \geq 1 \qquad \forall C_i \in C \tag{3.8}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \setminus \{(0, n+1)\} \tag{3.9}$$

$$y_i \in \{0, 1\} \qquad \forall i \in N \tag{3.10}$$

The objective function (3.1) is to maximize the total profit collected by the visited POIs. Constraints (3.2) and (3.3) are the degree constraints for the depots and the POI nodes, respectively. In particular, constraint (3.2) imposes that exactly m vehicles have to leave the origin depot and finish at the destination depot, whereas constraint (3.3) imposes that a POI node i is entered and left exactly once if it is visited. Constraint (3.4) bounds the flow originating from the origin depot. Constraint (3.5) updates the flow conservation value coming out from each visited POI. Constraints (3.6) and (3.7) set the time limits on the duration of each route, imposing lower and upper bounds on the values of the z variables in order to restrict the range of feasible values these variables can assume in fractional solutions. Note that if $x_{ij} = 1$, then z_{ij} denotes the arrival time at node j, whereas $x_{ij} = 0$ implies $z_{ij} = 0$. Constraint (3.8) ensures that for each cluster $C_i \in C$, at least one POI in the cluster is visited. Finally, constraints (3.9) and (3.10) are variable definition constraints.

This formulation has a polynomial number of variables and constraints. Specifically, the formulation given above has $O(n^2)$ binary decision variables and $O(n^2)$ constraints. In other words, the number of inequalities in these types of constraints grows polynomially with the number of nodes, i.e., they are polynomial-size formulations. Thus, this formulation remains manageable even as the problem size increases. Therefore, it can be used effectively to solve instances of small or moderate sizes by any optimizer.

4. Solving approach

In order to solve the TTDP-TC, we present an 'agile' biased-randomized iterated local search (BR-ILS) designed to produce high-quality solutions in short computing times. To achieve this, the proposed approach combines a new constructive heuristic tailored to the TTDP-TC, which generates high-quality initial solutions satisfying global type-coverage constraints, with a set of well-established metaheuristic components (e.g., perturbation operators, local search operators, and a demon-like acceptance criterion). While these components have individually proven effective for related routing problems, this is the first time they are adapted and orchestrated within a unified heuristic framework to efficiently solve the TTDP-TC. The resulting algorithm offers an effective balance between simplicity, solution quality, and computational efficiency, while requiring minimal parameter tuning [32].

Algorithm 1 outlines the steps of our BR-ILS solution approach. The algorithm begins with the generation of an initial solution using a biased-randomized version of an insertion heuristic. The insertion heuristic will be presented later in this section. The initial solution is then refined through a local search procedure that applies several local search operators to find the local minimum within the current neighborhood. This initial solution serves as the starting point and is assigned as both the base solution and the best solution found so far. Moreover, a credit variable is initialized to handle potential non-improving solutions. The ILS stage starts by iteratively exploring the search space within a given maximum time limit $maxTime$. During each iteration, a new solution is generated by applying a perturbation procedure to the current base solution. The perturbation process destroys a percentage of the base solution, and then reconstructs it using two operators. The new solution is then refined through the aforementioned local search procedure to find the local minimum within the current neighborhood. The perturbation and local search procedures will be described shortly. Next, the algorithm calculates the difference between the profit of the new solution and the current base solution. If the difference is negative, the base solution is updated with the new solution, and if the new solution is better than the best solution, the best solution is updated. If the new solution is worse, it can still be accepted under a demon-like acceptance criterion based on accumulated credit from previous large improvements that allows occasional acceptance of smaller deteriorations [33]. This mechanism promotes controlled exploration of the solution space and helps the search escape poor local optima without sacrificing long-term progress. The ILS process continues until the maximum allowed time is reached. Once the time limit is met, the best solution is returned as the output of the algorithm.

Algorithm 1 BR-ILS algorithm.

```
1: function BR-ILS($instance, maxTime, \beta, T_{\max}$)
2:     $initSol \leftarrow$ genInitSol($instance, \beta, T_{\max}$)          ▹ Initial solution stage
3:     $initSol \leftarrow$ localSearch($initSol, T_{\max}$)
4:     $baseSol \leftarrow initSol$
5:     $bestSol \leftarrow initSol$
6:     $credit \leftarrow 0$
7:     $elapsedTime \leftarrow 0$
8:     while $elapsedTime \leq maxTime$ do          ▹ ILS stage
9:         $newSol \leftarrow$ perturbation($baseSol, T_{\max}$)
10:        $newSol \leftarrow$ localSearch($newSol, T_{\max}$)
11:        $\Delta \leftarrow$ profit($baseSol$) $-$ profit($newSol$)
12:        if $\Delta < 0$ then
13:            $credit \leftarrow -\Delta$
14:            $baseSol \leftarrow newSol$
15:            if profit($newSol$) > profit($bestSol$) then
16:                $bestSol \leftarrow newSol$
17:            end if
18:        else          ▹ Demon-like acceptance criterion
19:            if $0 < \Delta \leq credit$ then
20:                $credit \leftarrow 0$
21:                $baseSol \leftarrow newSol$
22:            end if
23:        end if
24:        $elapsedTime \leftarrow$ updateTime()
25:    end while
26:    return $bestSol$
27: end function
```

Algorithm 2 outlines a two-stage biased-randomized insertion heuristic designed to generate high-quality and diverse initial solutions that already satisfy the global type-coverage constraints, while avoiding the infeasibility and bias typically produced by purely greedy construction. The first stage only considers the necessary POIs to ensure each cluster is visited, while the second stage tries to insert additional POIs into the solution. In the first stage, a dummy solution is created, where the m routes are defined. These routes start from the origin depot and arrive at the destination depot. Next, the heuristic inserts candidate POIs one at a time based on their regret values, inspired by [34]. In this context, the regret value of a POI is defined as the difference in travel times between its best and second-best insertion positions. The best position is determined as the one that minimizes the travel time when the POI is inserted into the solution. At each iteration, an insertion list with the regret values of each POI is calculated and sorted from highest to lowest, prioritizing POIs whose omission would result in the greatest regret. Next, an insertion element is selected in a biased-randomized manner. Biased-randomization techniques transform the deterministic insertion heuristic into a randomized algorithm while preserving its underlying logic [35]. Specifically, biased-randomization uses skewed probability distributions to induce a non-uniform random behavior. In this paper, a geometric probability distribution with a parameter $\beta \in (0, 1)$ is employed, controlling the relative level of greediness in the randomized behavior of the heuristic. Once a POI is inserted in its best position, it is removed from the candidate list along with all POIs that share the same cluster. This temporary removal ensures that POIs from other clusters are considered next, thereby promoting early coverage of all clusters and preventing the greedy insertion of multiple nearby POIs from the same cluster. If some clusters remain unvisited after this mechanism, γ POIs are removed from the solution and reinserted into the candidate list together with POIs from their corresponding clusters. This process is repeated within the first stage until all clusters are visited and the solution is coverage-feasible. Importantly, these removals are not intended as permanent exclusions: they are applied only during the first stage to enforce early cluster coverage and to mitigate excessive greediness that could otherwise bias the construction toward large or dense clusters. Next, the candidate list is updated with the remaining POIs not included in the solution, which are then considered for insertion in the second stage. This second stage considers all remaining POIs without any cluster-based exclusion (i.e., the temporary removals from the first stage are no longer applied), and biased-randomization is again used to avoid purely greedy behavior that could lead to systematically unbalanced solutions. In particular, POIs are inserted one at a time based on their preference values, defined as the product of their regret value and associated profit. At each iteration, an insertion list with the preference values of each POI is computed and sorted in descending order, after which a candidate is selected in a biased-randomized manner and inserted in its best position. This process is repeated until no further POIs can be inserted, and the resulting solution is returned.

Algorithm 3 details the perturbation procedure within the ILS stage, which intentionally destroys part of the current solution and reconstructs it using coverage-preserving operators to promote diversification while maintaining, or quickly restoring, coverage feasibility. This procedure is inspired by the destruction-construction procedure proposed by [36], where a portion of the base solution is destroyed and then reconstructed using two operators: a remove-insert operator and an exchange operator. The remove-insert operator removes POIs randomly from the solution γ_1 times. Then, it inserts the required POIs with the highest regret values back into the solution. The required POIs are those that need to be in the solution to visit each cluster. After all required POIs have been inserted, if the travel times are reduced, the obtained solution is accepted as the new solution. Otherwise, the base solution is returned as the new solution. Next, the exchange operator randomly selects pairs of POIs from different routes and exchanges their positions γ_2 times. If any routes violate the maximum time constraint ($T_{\max}$), POIs with the lowest preference values are removed until the solution becomes feasible (routes do not violate the maximum time constraint and visit each cluster), and the obtained solution is returned as the new solution. Otherwise, the base solution is returned as the new solution. Finally, the new solution is returned as the output of the perturbation procedure.

Algorithm 4 provides an overview of the local search procedure inspired by [36], which refines candidate solutions through a set of complementary intra-route and inter-route neighborhood operators designed to intensify the search by reducing travel times and improving POI allocation while preserving feasibility. The procedure starts with an active list to track the active POIs in the solution, which is initialized with all nodes in the current solution. The active list is then traversed in a random order, and for each active POI, we iterate through its candidate list of POIs. In this context, a candidate list for a specific POI includes only those POIs that can be reached within the maximum driving range ($T_{\max}$). For each pair of POIs, we encounter three distinct scenarios. *(i)* Both POIs belong to the same route, and a 2-opt operator is applied to try to reduce the route travel times. If the solution reduces the travel times, it is returned as the candidate solution. Otherwise, the current solution is returned and assigned to the candidate solution. *(ii)* Both POIs are part of two different routes, and an exchange operator followed by a relocation operator are employed to reduce the travel times of the solution. The exchange operator swaps the positions of both POIs, while the relocation operator removes the POIs and inserts them back at their optimal positions. The solution with the minimum travel times among the two operators is assigned as the candidate solution. *(ii)* The candidate POI has not been visited, and a relocation operator is employed to try to insert the candidate POI at its best position. If the solution is feasible, it is accepted and assigned as the candidate solution. Otherwise, an exchange operator is used to swap the positions of both POIs. If the solution remains feasible, it is accepted. Otherwise, the current solution is returned and assigned to the candidate solution. Next, if the candidate solution outperforms the current solution, both POIs and their adjacent left and right nodes in their routes are inserted into an auxiliary list. In addition, if the candidate solution outperforms the best solution, it is assigned as the best solution. After iterating through all arcs in the active list, the active list is replaced with the auxiliary list, and the best solution becomes the current solution. This process of searching for better solutions is repeated until the auxiliary list is empty. Finally, the best solution is returned as the output of the local search procedure.

Algorithm 2 Initial solution generation procedure.

```
 1: function genInitSol(instance, $\beta$, $T_{\max}$)
 2:     sol ← genDummySol(instance)                                   ▹ First stage
 3:     candidateList ← getNodes(instance)
 4:     while not isFeasible(sol) do
 5:         insertList ← calcRegretValues(sol, candidateList)
 6:         if size(insertList) ≠ 0 then
 7:             pos ← getRandomPosition(insertList, $\beta$)
 8:             index, route, node ← getInsertion(insertList, pos)
 9:             remove(candidateList, node)
10:             insert(route, node, index)
11:             nodes ← getClusterNodes(node)
12:             filter(candidateList, nodes)
13:         else
14:             nodesToRemove ← $\emptyset$
15:             for $i = 1$ to $\gamma$ do
16:                 route ← getRandomNode(sol)
17:                 node ← getRandomNode(route)
18:                 remove(route, node)
19:                 insert(nodesToRemove, node)
20:             end for
21:             for node ∈ nodesToRemove do
22:                 nodes ← getClusterNodes(node)
23:                 insert(candidateList, nodes)
24:             end for
25:         end if
26:     end while
27:     isFeasible ← true                                             ▹ Second stage
28:     candidateList ← getNodes(instance)
29:     filter(candidateList, getNodes(sol))
30:     while isFeasible do
31:         insertList ← calcPreferenceValues(sol, candidateList)
32:         if size(insertList) ≠ 0 then
33:             pos ← getRandomPosition(insertList, $\beta$)
34:             index, route, node ← getInsertion(insertList, pos)
35:             remove(candidateList, node)
36:             insert(route, node, index)
37:         else
38:             isFeasible ← false
39:         end if
40:     end while
41:     return sol
42: end function
```

Algorithm 3 Perturbation procedure.

```
1: function PERTURBATION($baseSol, T_{\max}$)
2:     $newSol \leftarrow$ REMOVEINSERTOPT($baseSol, T_{\max}$)
3:     $newSol \leftarrow$ EXCHANGEOPT($newSol, T_{\max}$)
4:     return $newSol$
5: end function
```

Algorithm 4 Local search procedure.

```
 1: function LOCALSEARCH($newSol, T_{\max}$)
 2:     $bestSol \leftarrow newSol$
 3:     $currentSol \leftarrow newSol$
 4:     $activeList \leftarrow$ GETNODES($currentSol$)
 5:     while SIZE($activeList$) $\neq 0$ do
 6:         $auxiliaryList \leftarrow \emptyset$
 7:         SHUFFLE($activeList$)
 8:         for $iNode \in activeList$ do
 9:             for $jNode \in$ CANDIDATELIST($iNode$) do
10:                 $pos \leftarrow$ GETRELATIVEPOSITION($currentSol, iNode, jNode$)
11:                 if $pos = 1$ then
12:                     $candSol \leftarrow$ TWOOPT($currentSol, iNode, jNode, T_{\max}$)
13:                 else if $pos = 2$ then
14:                     $exchangeSol \leftarrow$ EXCHANGEOPT($currentSol, iNode, jNode, T_{\max}$)
15:                     $relocateSol \leftarrow$ RELOCATIONOPT($currentSol, iNode, jNode, T_{\max}$)
16:                     if TRAVELTIME($exchangeSol$) $<$ TRAVELTIME($relocateSol$) then
17:                         $candSol \leftarrow exchangeSol$
18:                     else
19:                         $candSol \leftarrow relocateSol$
20:                     end if
21:                 else if $pos = 3$ then
22:                     $relocateSol \leftarrow$ RELOCATIONOPT2($currentSol, iNode, jNode, T_{\max}$)
23:                     if PROFIT($relocateSol$) $>$ PROFIT($candSol$) then
24:                         $candSol \leftarrow relocateSol$
25:                     else
26:                         $candSol \leftarrow$ EXCHANGEOPT2($currentSol, iNode, jNode, T_{\max}$)
27:                     end if
28:                 end if
29:                 if PROFIT($candSol$) $>$ PROFIT($currentSol$) then
30:                     $iLeft, iRight \leftarrow$ GETLEFTRIGHT($currentSol, iNode$)
31:                     INSERT($auxiliaryList, iNode, iLeft, iRight$)
32:                     $jLeft, jRight \leftarrow$ GETLEFTRIGHT($currentSol, jNode$)
33:                     INSERT($auxiliaryList, jNode, jLeft, jRight$)
34:                 end if
35:                 if PROFIT($candSol$) $>$ PROFIT($bestSol$) then
36:                     $bestSol \leftarrow candSol$
37:                 end if
38:             end for
39:         end for
40:         $currentSol \leftarrow bestSol$
41:         $activeList \leftarrow auxiliaryList$
42:     end while
43:     return $bestSol$
44: end function
```

5. Computational experiments

The computational experiments were conducted on a Linux Mint machine with a multi-core processor Intel Xeon E5-2630 v4 with 32 GB of RAM. The formal model described in Section 3 was implemented using Python 3.11 (`https://www.python.org/`, accessed on January 26, 2026) and solved with the commercial solver Gurobi to calculate the optimal values for different benchmark instances. A maximum time limit of 2 hours was set for the exact solver to find optimal solutions. This way, the performance of the proposed agile algorithm can be compared to the mathematical approach. Similarly, the algorithm presented in Section 4 was implemented in Julia 1.10 (`https://julialang.org/`, accessed on January 26, 2026). For the proposed algorithm, a maximum of 60 seconds was used as the stopping criterion. For the biased-randomization parameter, we set it to be randomly selected in the interval $(0.3, 0.4)$ after a quick tuning process over a random sample of instances which established a good performance. Each instance was executed 32 times with different seeds for the random number generator using different threads of the workstation, and the obtained results are reported.

To the best of our knowledge, the TTDP-TC has never been studied previously in the literature. Thus, there are no existing benchmark instances to test our solving methodology. Accordingly, we extend the instances proposed by [37] for the TOP to assess both the performance and quality of the proposed model and algorithm (Table 2). The process involves assigning a cluster to each of the POIs following the rules proposed by [22]. Specifically, the number k of clusters was set to the following values: 5, 10, 15, 20, 25. Clusters are generated by considering the vertices sequentially with respect to their identifier and inserting them in each cluster in such a way as to obtain clusters of similar size, and which are partially overlapping. In particular, we start by removing the first and last vertices (depots) from the set V and obtain the list of n POIs. Then, we compute the quotient q and the remainder r of the integer division $n \div k$. The POIs list is thus partitioned into k clusters where the first r clusters contain $q + 1$ POIs, and the remaining $k - r$ clusters contain q POIs. Finally, we compute $d = \max\left(1, \left\lfloor \frac{q}{10} \right\rfloor\right)$, and make each cluster share its first d POIs with the preceding cluster and its last d POIs with the following one (the last cluster is the preceding cluster of the first one and vice versa). As a final result, two consecutive clusters overlap for $2d$ POIs, i.e., each cluster shares POIs with two other clusters for a total of $4d$ POIs.

Table 2. Summary of benchmark instances.

Set	#	$\lvert V\rvert$	m	$T_{\max}$
1	54	32	2–4	3.8–22.5
2	33	21	2–4	1.2–42.5
3	60	33	2–4	3.8–55.0
4	60	100	2–4	3.8–40.0
5	78	66	2–4	1.2–65.0
6	42	64	2–4	5.0–200.0
7	60	102	2–4	12.5–120.0

The benchmark instances described here can be found at `https://www.researchgate.net/publication/385343592_Benchmark_instances_for_the_TTDP-TC`. In addition, the best-known solutions and the upper bounds obtained by the solver along with the best-found solutions using the BR-ILS algorithm are reported.

6. Analysis of results

Tables 3–5 present a summary of the obtained results for the benchmark instances for different numbers of vehicles. Each table organizes the instances based on the benchmark set (rows) and the number of clusters k (columns). For each set of instances, the following information is reported:

- Number of instances (#).
- Number of instances solved to optimality by the solver (# Solved).
- Number of optimal solutions found by our BR-ILS algorithm (# Opt).
- Average percentage gap and average computational time of the best-known solutions obtained by the solver compared to the upper bounds (Av. Gap and Av. Time).
- Average percentage gap and average computational time of the best found solutions obtained by the BR-ILS algorithm compared to the upper bounds computed by the solver (Av. Gap$*$ and Av. Time$*$).

As shown in Tables 3–5, the first three sets comprise the simplest instances, as both the solver and the BR-ILS algorithm find the optimal solutions for all instances. On the other hand, the rest of the sets consist of instances more difficult to solve to optimality. This is inherited by the characteristics of TOP instances. Moreover, the performance of both the solver and BR-ILS algorithm are influenced by the number of clusters as the number of instances solved to optimality decreases as the number of clusters k increases. In other words, as the instances have a greater number of clusters, less profit is collected in favor of visiting more clusters, which limits the number of valid configurations of nodes to visit in order to ensure cluster coverage. Lastly, it can be noted that the performance also varies with the number of vehicles m. The solver and algorithm are particularly effective for $m = 2$, achieving more optimal solutions and lower average gaps. As the number of vehicles increases, the number of optimal solutions decreases and the percentage gaps increase when considering the upper bound values as reference. This underscores the strengths and disadvantages of both approaches.

A closer analysis of these obtained results reveals more details. If we compare the average percentage gaps for the solver and BR-ILS algorithm, it is observed that sometimes the average gap is lower for the proposed algorithm. In other words, the BR-ILS algorithm is able to outperform the best-known solutions obtained by the solver. In particular, we can highlight the average percentage gaps obtained for the fifth set with $m = 3$. The solver achieves average gaps ranging from 0.58 to 0.75, while the algorithm obtains average gaps ranging from 0.24 to 0.39 for the different numbers of clusters. This shows that our approach is an effective algorithm to solve the TTDP-TC. Moreover, these solutions are achieved with significantly lower average computational times. Specifically, if we compare the computational time of the BR-ILS algorithm with respect to the solver, it is evident that the algorithm achieves results significantly faster. The average computational times for the solver range from 2872.33 to 4684.17 seconds, while the algorithm maintains much lower times, averaging between 22.47 and 39.36 seconds. Despite this substantial reduction in computation time, the algorithm's solution quality remains close to that of the solver, with average percentage gaps that do not increase greatly. In particular, the solver obtains average gaps ranging from 0.28 to 2.21, while the algorithm obtains gaps ranging from 0.44 to 1.44. Hence, the solution quality of the algorithm closely approximates that of the solver, indicating that it efficiently balances computational speed with solution accuracy.

Table 3. Computational results for $m = 2$.

Set		$k = 5$	$k = 10$	$k = 15$	$k = 20$	$k = 25$
	#	15	13	12	10	9
	# Solved	15	13	12	10	9
	# Opt	15	13	12	10	9
1	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	1.67	1.72	2.32	2.45	2.18
	Av. Gap∗	0.00	0.00	0.00	0.00	0.00
	Av. Time∗	10.22	16.18	3.64	11.88	2.58
	#	9	2	1	1	-
	# Solved	9	2	1	1	-
	# Opt	9	2	1	1	-
2	Av. Gap	0.00	0.00	0.00	0.00	-
	Av. Time	0.26	0.55	0.94	1.35	-
	Av. Gap∗	0.00	0.00	0.00	0.00	-
	Av. Time∗	0.07	0.03	1.58	28.32	-
	#	19	16	14	14	14
	# Solved	19	16	14	14	14
	# Opt	19	16	14	14	14
3	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	2.98	4.79	3.77	3.64	3.45
	Av. Gap∗	0.00	0.00	0.00	0.00	0.00
	Av. Time∗	7.25	12.61	13.79	15.11	6.61
	#	20	20	19	18	18
	# Solved	19	20	19	18	17
	# Opt	4	5	5	2	1
4	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	660.01	544.32	566.29	479.39	714.59
	Av. Gap∗	1.88	1.76	1.65	1.88	1.99
	Av. Time∗	12.83	27.41	27.93	34.00	35.03
	#	22	21	21	19	17
	# Solved	22	21	21	19	17
	# Opt	15	12	15	10	9
5	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	42.94	36.03	36.90	53.65	45.51
	Av. Gap∗	0.32	0.48	0.30	0.55	0.59
	Av. Time∗	18.26	26.82	19.71	14.52	19.81
	#	11	11	11	10	10
	# Solved	11	11	11	10	10
	# Opt	9	11	11	10	9
6	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	58.07	35.86	43.06	70.72	50.72
	Av. Gap∗	0.09	0.00	0.00	0.00	0.11
	Av. Time∗	17.11	13.53	20.62	18.93	15.77
	#	18	17	15	14	12
	# Solved	18	17	15	14	12
	# Opt	6	7	1	1	1
7	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	104.60	110.84	193.31	153.93	404.93
	Av. Gap∗	1.52	1.07	2.16	2.02	2.67
	Av. Time∗	34.26	29.06	34.63	40.47	34.50

Table 4. Computational results for $m = 3$.

Set		$k = 5$	$k = 10$	$k = 15$	$k = 20$	$k = 25$
	#	13	11	10	7	6
	# Solved	13	11	10	7	6
	# Opt	13	11	10	7	6
1	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	1.52	1.71	2.08	4.58	2.24
	Av. Gap$*$	0.00	0.00	0.00	0.00	0.00
	Av. Time$*$	1.98	3.95	19.71	1.34	2.98
	#	5	-	-	-	-
	# Solved	5	-	-	-	-
	# Opt	5	-	-	-	-
2	Av. Gap	0.00	-	-	-	-
	Av. Time	0.16	-	-	-	-
	Av. Gap$*$	0.00	-	-	-	-
	Av. Time$*$	0.00	-	-	-	-
	#	18	13	10	10	10
	# Solved	18	13	10	10	10
	# Opt	18	13	10	10	10
3	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	22.29	22.80	33.72	45.75	33.01
	Av. Gap$*$	0.00	0.00	0.00	0.00	0.00
	Av. Time$*$	2.43	8.02	10.93	4.20	4.23
	#	18	18	17	16	16
	# Solved	14	14	13	12	13
	# Opt	4	4	3	2	3
4	Av. Gap	0.19	0.16	0.16	0.13	0.15
	Av. Time	2070.30	2047.16	2275.55	2316.68	2131.49
	Av. Gap$*$	2.19	1.96	2.03	2.04	1.81
	Av. Time$*$	23.98	32.46	32.64	35.02	41.63
	#	20	18	18	17	15
	# Solved	14	12	12	11	10
	# Opt	11	9	9	7	6
5	Av. Gap	0.64	0.73	0.58	0.75	0.64
	Av. Time	2653.25	2949.22	3001.97	3098.46	3090.19
	Av. Gap$*$	0.30	0.24	0.26	0.35	0.39
	Av. Time$*$	24.31	24.74	21.00	23.44	17.44
	#	8	8	8	7	7
	# Solved	8	8	8	7	7
	# Opt	8	8	8	7	7
6	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	225.85	198.94	229.13	620.65	430.97
	Av. Gap$*$	0.00	0.00	0.00	0.00	0.00
	Av. Time$*$	21.35	11.26	28.65	25.48	13.34
	#	17	15	14	12	10
	# Solved	15	12	11	9	8
	# Opt	7	5	3	2	0
7	Av. Gap	0.21	0.25	0.31	0.45	0.39
	Av. Time	1606.86	1875.35	1981.10	2408.44	2370.21
	Av. Gap$*$	1.16	1.02	0.78	1.80	2.65
	Av. Time$*$	28.11	27.94	23.43	41.98	30.43

Table 5. Computational results for $m = 4$.

Set		$k = 5$	$k = 10$	$k = 15$	$k = 20$	$k = 25$
	#	12	9	8	2	2
	# Solved	12	9	8	2	2
	# Opt	12	9	8	2	2
1	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	0.64	0.90	0.85	8.45	2.58
	Av. Gap∗	0.00	0.00	0.00	0.00	0.00
	Av. Time∗	0.32	0.83	5.44	0.42	0.48
	#	1	-	-	-	-
	# Solved	1	-	-	-	-
	# Opt	1	-	-	-	-
2	Av. Gap	0.00	-	-	-	-
	Av. Time	0.17	-	-	-	-
	Av. Gap∗	0.00	-	-	-	-
	Av. Time∗	0.00	-	-	-	-
	#	16	10	6	6	6
	# Solved	16	10	6	6	6
	# Opt	16	10	6	6	6
3	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	22.98	48.45	52.07	52.05	74.01
	Av. Gap∗	0.00	0.00	0.00	0.00	0.00
	Av. Time∗	1.25	5.16	0.59	3.13	0.95
	#	16	16	15	14	14
	# Solved	8	9	9	7	6
	# Opt	5	5	4	4	3
4	Av. Gap	1.05	0.97	0.71	0.89	1.03
	Av. Time	3672.34	3546.51	3701.18	3907.69	4228.30
	Av. Gap∗	1.47	1.49	1.82	1.88	1.58
	Av. Time∗	36.92	29.70	33.93	34.78	35.05
	#	18	15	15	15	13
	# Solved	12	10	9	10	9
	# Opt	11	8	7	8	6
5	Av. Gap	0.58	0.72	0.77	0.52	0.63
	Av. Time	2971.81	3195.65	3326.03	2760.75	2698.66
	Av. Gap∗	0.52	0.57	0.47	0.68	0.63
	Av. Time∗	21.40	31.76	24.65	20.73	18.42
	#	5	5	5	4	4
	# Solved	5	5	5	4	4
	# Opt	5	5	5	4	4
6	Av. Gap	0.00	0.00	0.00	0.00	0.00
	Av. Time	7.18	7.40	6.89	7.54	8.51
	Av. Gap∗	0.00	0.00	0.00	0.00	0.00
	Av. Time∗	23.64	17.20	17.57	34.80	22.89
	#	16	14	13	10	9
	# Solved	12	9	7	7	5
	# Opt	7	6	4	2	0
7	Av. Gap	0.31	0.43	0.72	0.28	2.21
	Av. Time	2872.33	3447.99	3874.34	3494.00	4684.17
	Av. Gap∗	0.73	0.44	0.68	0.93	1.44
	Av. Time∗	22.47	23.85	28.62	39.36	28.69

Figures 2–4 provides a comprehensive overview of the performance of our algorithm across different numbers of vehicles, with respect to the best-known solutions found by the solver. Notably, our BR-ILS algorithm obtains solutions comparable to those of the solver, achieving average gaps ranging from 0.64 to 0.99 for $m = 2$, from 0.66 to 0.96 for $m = 3$, and from 0.53 to 0.90 for $m = 4$. This trend suggests that increasing the number of vehicles allows the algorithm to obtain solutions closer to those of the solver. Thus, this discrepancy suggests that the upper bound obtained through the solver may deviate significantly from the optimal solution. In other words, as the number of vehicles increases the performance of the solver decreases and less optimal solutions are found. These results

highlight the quality of our proposed heuristic method compared to the solver. Another detail supported by the previous analysis is the obtention of solutions that outperform the ones achieved by the solver. In particular, we highlight the minimum value of the boxplot for $m = 3$ and $k = 5$ being below the best-known solution values obtained by the solver. This further establishes the competitiveness of the BR-ILS algorithm.

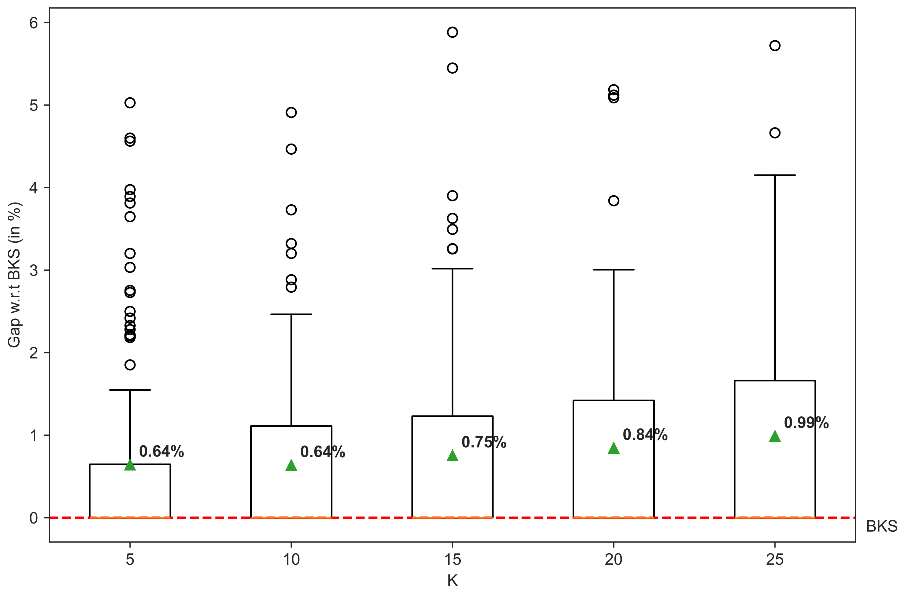

Figure 2. Percentage gaps using our BR-ILS algorithm for $m = 2$.

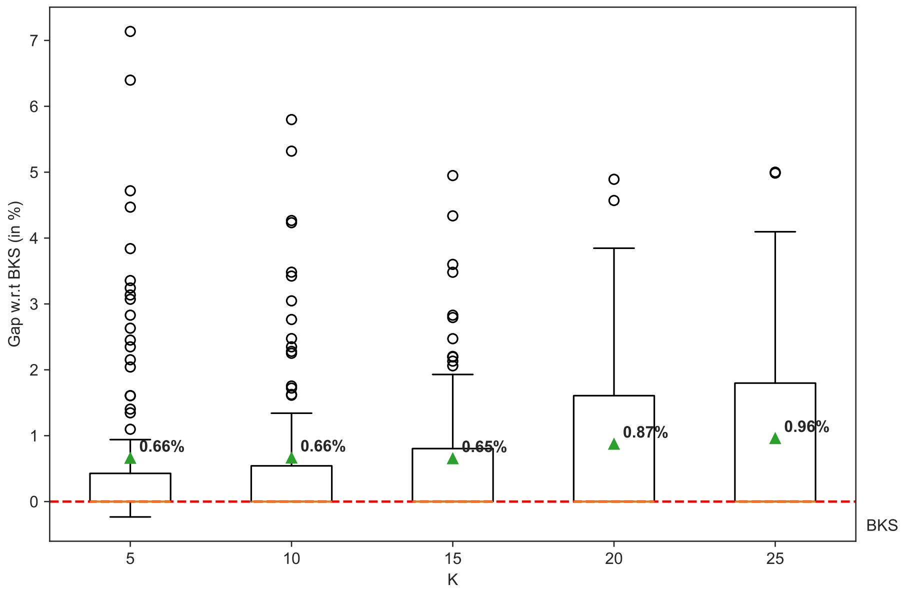

Figure 3. Percentage gaps using our BR-ILS algorithm for $m = 3$.

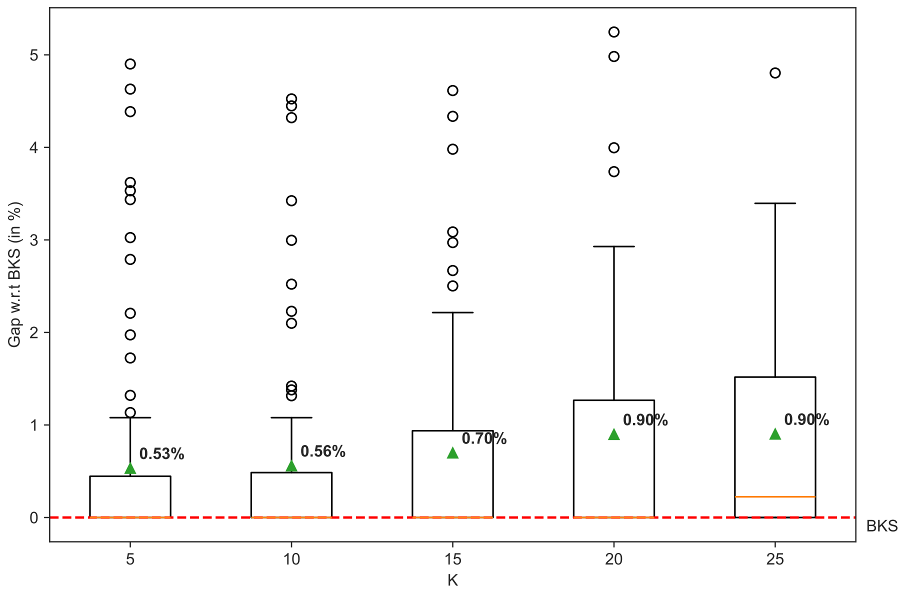

Figure 4. Percentage gaps using our BR-ILS algorithm for $m = 4$.

To illustrate the practical applicability of our BR-ILS algorithm, we consider a real-world illustrative instance based on tourism data from the Barcelona Open Data repository (`https://opendata-ajuntament.barcelona.cat/`). The instance consists of 877 real touristic POIs with geographic coordinates and category labels (e.g., cultural attractions, leisure activities, gastronomy), which are mapped to clusters required by the TTDP-TC. Travel times between POIs are derived from their geographic coordinates using Haversine distances, and profit values are assigned based on publicly available popularity indicators. Moreover, we impose a maximum travel limit of $T_{\max} = 10$ km, reflecting the typical daily walking distance of an average urban tourist, and we define the origin and destination depots at the city center, computed as the centroid of the considered POI coordinates. Lastly, we consider itineraries of 2, 3, and 4 days to reflect different realistic trip planning horizons. Figure 5 shows the spatial distribution of the considered POIs across the city of Barcelona, colored according to their assigned clusters. This highlights both the geographical dispersion of POIs and the heterogeneity of clusters within the urban environment.

Table 6 summarizes the computational performance of the BR-ILS algorithm on the Barcelona Open Data instance for planning horizons of 2, 3, and 4 days, using the same experimental configuration described in Section 5. For each planning horizon, the table distinguishes between the best solution obtained and the average performance across all runs, reporting the total collected profit, number of visited POIs, CPU time to best-found solution, and the distribution of POIs among the five clusters. Moreover, Gurobi was not considered for comparison as it is unable to handle this large-scale instance, further highlighting the practical relevance of heuristic approaches. Notice that, as expected, increasing the number of days leads to a substantial increase in both the total profit and the number of visited POIs. Specifically, the best collected profit grows from 1301 to 1781 and 2289, while the corresponding average profits increase from 900.3 to 1434.2 and 1717.9. In addition, the cluster composition of both the best and average solutions are dominated by leisure and cultural POIs, followed by gastronomy, while nature and shopping POIs appear less frequently, reflecting both their relative availability and the profit structure of the instance. Overall, the results indicate that BR-ILS scales well with the planning horizon, consistently producing high-quality, coverage-feasible itineraries, and exhibiting stable performance across multiple independent runs.

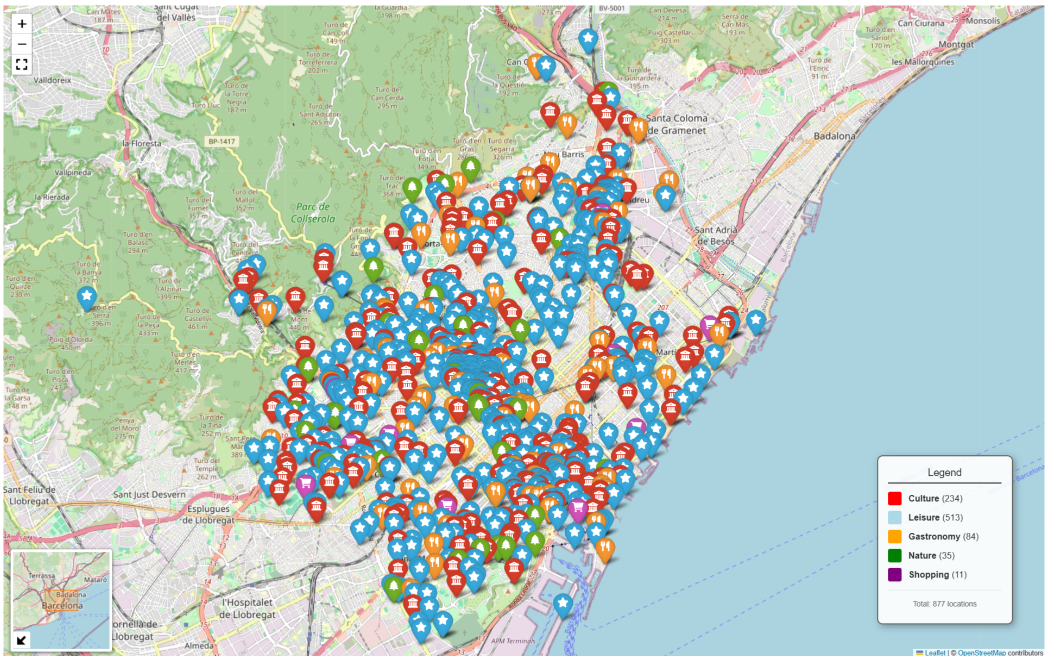

Figure 5. Spatial distribution of touristic POIs in Barcelona grouped in 5 clusters.

Table 6. Computational results of BR-ILS on the Barcelona Open Data instance.

	Best				Average			
Days	Profit	Nodes	Time (s)	Clusters (C1–C5)	Profit	Nodes	Time (s)	Clusters (C1–C5)
2	1,301	124	18.1	33 / 74 / 14 / 2 / 1	900.3	90.6	10.8	24.6 / 52.5 / 10.2 / 2.4 / 1.2
3	1,781	178	28.9	44 / 103 / 21 / 9 / 1	1,434.2	138.9	29.5	36.2 / 75.7 / 15.0 / 4.4 / 1.6
4	2,289	217	49.5	64 / 114 / 30 / 8 / 1	1,717.9	163.4	50.6	43.9 / 88.9 / 17.5 / 5.5 / 1.8

7. Conclusions

This paper analyzes a complex and realistic version of the TTDP involving type-covering constraints. Instead of focusing solely on maximizing the profit from visiting POIs, a more comprehensive approach is adopted. The concept of type coverage is introduced, where each route must include at least one POI from each specified type, such as museums, leisure activities, restaurants, and bars. This constraint introduces additional complexity to the optimization problem and its solution approach.

Given the need to solve instances of this problem within short computational times, since tourist plans often need to be adjusted on the fly, an agile optimization algorithm is proposed. Specifically, we present an ‘agile’ BR-ILS metaheuristic designed to efficiently handle the TTDP-TC. To validate the competitiveness of our approach, we implement an exact method and solve it with the commercial solver Gurobi. Our computational experiments demonstrate the efficacy of our approach, highlighting its ability to generate high-quality solutions within practical time frames. This confirms the practical viability and robustness of the proposed method in optimizing tourist route planning while ensuring diverse POI coverage.

Several future research lines can be explored as follows: *(i)* consider extending the TTDP-TC to include time-dependent POI availability, where opening hours and peak times affect route planning; *(ii)* explore a richer variant of the TTDP-TC that incorporates tourist-specific or non-linear preference structures, such as diminishing returns, lexicographic preferences, or personalized utility functions, allowing the model to capture more realistic satisfaction dynamics beyond a linear profit formulation; *(iii)* consider balancing the distribution of POI types across individual routes, avoiding routes dominated by a single type to improve the daily tourist experience; *(iv)* study soft type-covering formulations, where coverage requirements are relaxed through penalties, weights, or lexicographic rules, allowing trade-offs between profit maximization and category diversity, thus better reflecting flexible tourist preferences; and *(v)* develop a stochastic version of the problem where POI profits or travel times between POIs are modeled as random variables, potentially addressed by extending our metaheuristic into a simheuristic approach.

Author contributions

Xabier A. Martin: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft. Javier Panadero: Software, Validation, Writing – review & editing. Angel A. Juan: Conceptualization, Supervision, Validation, Writing – review & editing.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Conflict of interest

The authors declare that they have no competing interests.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science (PID2022-138860NB-I00 and RED2022-134703-T).

References

1. J. Ruiz-Meza, J. R. Montoya-Torres, A systematic literature review for the tourist trip design problem: Extensions, solution techniques and future research lines, *Oper. Res. Perspect.*, **9** (2022), 100228. https://doi.org/10.1016/j.orp.2022.100228

2. P. Vansteenwegen, W. Souffriau, Trip planning functionalities: State of the art and future, *Inf. Technol. Tour.*, **12** (2010), 305–315. https://doi.org/10.3727/109830511X13049763021853

3. T. Tsiligirides, Heuristic methods applied to orienteering, *J. Oper. Res. Soc.*, **35** (1984), 797–809. https://doi.org/10.1057/jors.1984.162

4. R. Gama, H. L. Fernandes, A reinforcement learning approach to the orienteering problem with time windows, *Comput. Oper. Res.*, **133** (2021), 105357. https://doi.org/10.1016/j.cor.2021.105357

5. S. W. Lin, V. F. Yu, Solving the team orienteering problem with time windows and mandatory visits by multi-start simulated annealing, *Comput. Ind. Eng.*, **114** (2017), 195–205. https://doi.org/10.1016/j.cie.2017.10.020

6. M. Khodadadian, A. Divsalar, C. Verbeeck, A. Gunawan, P. Vansteenwegen, Time dependent orienteering problem with time windows and service time dependent profits, *Comput. Oper. Res.*, **143** (2022), 105794. https://doi.org/10.1016/j.cor.2022.105794

7. H. Kim, B. I. Kim, D. jin Noh, The multi-profit orienteering problem, *Comput. Ind. Eng.*, **149** (2020), 106808. https://doi.org/10.1016/j.cie.2020.106808

8. C. Bayliss, A. A. Juan, C. S. Currie, J. Panadero, A learnheuristic approach for the team orienteering problem with aerial drone motion constraints, *Appl. Soft Comput.*, **92** (2020), 106280. https://doi.org/10.1016/j.asoc.2020.106280

9. A. Estrada-Moreno, A. Ferrer, A. A. Juan, J. Panadero, A. Bagirov, The non-smooth and bi-objective team orienteering problem with soft constraints, *Mathematics*, **8** (2020), 1461. https://doi.org/10.3390/math8091461

10. L. Evers, K. Glorie, S. van der Ster, A. I. Barros, H. Monsuur, A two-stage approach to the orienteering problem with stochastic weights, *Comput. Oper. Res.*, **43** (2014), 248–260. https://doi.org/10.1016/j.cor.2013.09.011

11. J. Panadero, A. A. Juan, C. Bayliss, C. Currie, Maximising reward from a team of surveillance drones: a simheuristic approach to the stochastic team orienteering problem, *Eur. J. Ind. Eng.*, **14** (2020), 485. https://doi.org/10.1504/EJIE.2020.108581

12. J. Panadero, M. Ammouriova, A. A. Juan, A. Agustin, M. Nogal, C. Serrat, Combining parallel computing and biased randomization for solving the team orienteering problem in real-time, *Appl. Sci.*, **11** (2021), 12092. https://doi.org/10.3390/app112412092

13. T. İlhan, S. M. R. Iravani, M. S. Daskin, The orienteering problem with stochastic profits, *IIE Trans.*, **40** (2008), 406–421. https://doi.org/10.1080/07408170701592481

14. P. Vansteenwegen, W. Souffriau, D. V. Oudheusden, The orienteering problem: A survey, *Eur. J. Oper. Res.*, **209** (2011), 1–10. https://doi.org/10.1016/j.ejor.2010.03.045

15. A. Gunawan, H. C. Lau, P. Vansteenwegen, Orienteering problem: A survey of recent variants, solution approaches and applications, *Eur. J. Oper. Res.*, **255** (2016), 315–332. https://doi.org/10.1016/j.ejor.2016.04.059

16. C. Archetti, F. Carrabs, R. Cerulli, The set orienteering problem, *Eur. J. Oper. Res.*, **267** (2018), 264–272. https://doi.org/10.1016/j.ejor.2017.11.009

17. F. Carrabs, A biased random-key genetic algorithm for the set orienteering problem, *Eur. J. Oper. Res.*, **292** (2021), 830–854. https://doi.org/10.1016/j.ejor.2020.11.043

18. R. Pěnička, J. Faigl, M. Saska, Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants, *Eur. J. Oper. Res.*, **276** (2019), 816–825. https://doi.org/10.1016/j.ejor.2019.01.047
19. M. Dontas, G. Sideris, E. G. Manousakis, An adaptive memory matheuristic for the set orienteering problem, *Eur. J. Oper. Res.*, (2023). https://doi.org/10.1016/j.ejor.2023.02.008
20. T. D. Nguyen, R. Martinelli, Q. A. Pham, M. H. Hà, The set team orienteering problem, *Eur. J. Oper. Res.*, **321** (2025), 75–87. https://doi.org/10.1016/j.ejor.2024.09.021
21. A. Gunawan, V. F. Yu, A. N. Sutanto, P. Jodiawan, Set team orienteering problem with time windows, in *Learning and Intelligent Optimization: 15th International Conference, LION 15, Athens, Greece, June 20–25, 2021, Revised Selected Papers 15*. Springer, 2021, 142–149. https://doi.org/10.1007/978-3-030-92121-7_12
22. E. Angelelli, C. Archetti, M. Vindigni, The clustered orienteering problem, *Eur. J. Oper. Res.*, **238** (2014), 404–414. https://doi.org/10.1016/j.ejor.2014.04.006
23. A. E. Yahiaoui, A. Moukrim, M. Serairi, Hybrid heuristic for the clustered orienteering problem, in *Computational Logistics: 8th International Conference, ICCL 2017, Southampton, UK, October 18-20, 2017, Proceedings 8*. Springer, 2017, 19–33. https://doi.org/10.1007/978-3-319-68496-3_2
24. Q. Wu, M. He, J. K. Hao, Y. Lu, An effective hybrid evolutionary algorithm for the clustered orienteering problem, *Eur. J. Oper. Res.*, **313** (2024), 418–434. https://doi.org/10.1016/j.ejor.2023.08.006
25. A. E. Yahiaoui, A. Moukrim, M. Serairi, The clustered team orienteering problem, *Comput. Oper. Res.*, **111** (2019), 386–399. https://doi.org/10.1016/j.cor.2019.07.008
26. T. Derya, E. Dinler, B. Keçeci, Selective generalized travelling salesman problem, *Math. Comput. Model. Dynam. Syst.*, **26** (2020), 80–118. https://doi.org/10.1080/13873954.2019.1705496
27. K. Ameranis, N. Vathis, D. Fotakis, Minimum and maximum category constraints in the orienteering problem with time windows, in *International Symposium on Experimental Algorithms*. Springer, 2019, 343–358. https://doi.org/10.1007/978-3-030-34029-2_23
28. C. Panagiotakis, E. Daskalaki, H. Papadakis, P. Fragopoulou, An expectation-maximization framework for personalized itinerary recommendation with poi categories and must-see pois, *ACM Trans. Recom. Syst.*, **3** (2024), 1–33. https://doi.org/10.1145/3696114
29. A. Expósito, S. Mancini, J. Brito, J. A. Moreno, A fuzzy grasp for the tourist trip design with clustered pois, *Expert Syst. Appl.*, **127** (2019), 210–227. https://doi.org/10.1016/j.eswa.2019.03.004
30. D. Jriji, S. Krichen, F. Madany, A memetic algorithm for the tourist trip design with clustered points of interests, in *2020 International Multi-Conference on:"Organization of Knowledge and Advanced Technologies"(OCTA)*. IEEE, 2020, 1–6. https://doi.org/10.1109/OCTA49274.2020.9151767

31. N. Bianchessi, R. Mansini, M. G. Speranza, A branch-and-cut algorithm for the team orienteering problem, *Int. T. Oper. Res.*, **25** (2018), 627–635. https://doi.org/10.1111/itor.12422

32. H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search: Framework and applications, in *Handbook of Metaheuristics*, Springer, 2018, 129–168. https://doi.org/10.1007/978-3-319-91086-4_5

33. E. G. Talbi, *Metaheuristics: from design to implementation*, John Wiley & Sons, 2009. https://doi.org/10.1002/9780470496916

34. S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, *Transport. Sci.*, **40** (2006), 455–472. https://doi.org/10.1287/trsc.1050.0135

35. A. A. Juan, P. Keenan, R. Martí, S. McGarraghy, J. Panadero, P. Carroll, et al., A review of the role of heuristics in stochastic optimisation: From metaheuristics to learnheuristics, *Ann. Oper. Res.*, **320** (2023), 831–861. https://doi.org/10.1007/s10479-021-04142-9

36. L. Ke, W. Yang, Reformulation and metaheuristic for the team orienteering arc routing problem, in *Advances in Swarm Intelligence: 8th International Conference, ICSI 2017, Fukuoka, Japan, July 27–August 1, 2017, Proceedings, Part II 8*. Springer, 2017, 494–501. https://doi.org/10.1007/978-3-319-61833-3_52

37. I. M. Chao, B. L. Golden, E. A. Wasil, The team orienteering problem, *Eur. J. Oper. Res.*, **88** (1996), 464–474. https://doi.org/10.1016/0377-2217(94)00289-4

AIMS Press