Journal of Combinatorial Optimization (2023) 46:1
https://doi.org/10.1007/s10878-023-01066-x

New mixed integer linear programming models and an iterated local search for the clustered traveling salesman problem with relaxed priority rule

Thanh Tan Doan[1,2] · Nathalie Bostel[1] · Minh Hoàng Hà[2] · Vu Hoang Vuong Nguyen[3]

Accepted: 4 July 2023 / Published online: 24 July 2023

Abstract
The Traveling Salesman Problem (TSP) is a well known problem in operations research with various studies and applications. In this paper, we address a variant of the TSP in which the customers are divided into several priority groups and the order of servicing groups can be flexibly changed with a rule called the d-relaxed priority rule. The problem is called the Clustered Traveling Salesman Problem with Relaxed Priority Rule (CTSP-d). We propose two new Mixed Integer Linear Programming (MILP) models for the CTSP-d and a metaheuristic based on Iterated Local Search (ILS) with operators designed for or adapted to the problem. The experimental results obtained on the benchmark instances show that two new models performs better than previous ones, and ILS also proves its performance with 13 new best known solutions found and significant stability compared to existing metaheuristics.

Keywords Traveling Salesman Problem · d-relaxed priority rule · Mixed integer linear programming · Iterated local search

1 Introduction

The Traveling Salesman Problem (TSP) is a classical routing problem where many approaches, applications and solution methods can be found in surveys such as (Applegate et al. 2006; Gutin and Punnen 2007), and Matai et al. (2010). Our study addresses

✉ Minh Hoàng Hà
hoang.haminh@phenikaa-uni.edu.vn

1 CNRS, LS2N, Université de Nantes, 44000 Nantes, France

2 ORLab, Faculty of Computer Science, Phenikaa University, Hanoi 12116, Vietnam

3 ORLab, Faculty of Information Technology, VNU University of Engineering and Technology, Hanoi, Vietnam

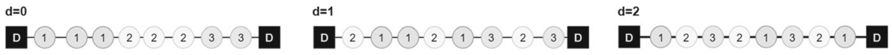

Fig. 1 CTSP-d solutions with 3 priority groups and different values of d

a TSP variant in which nodes are divided into several priority groups and the order of service can be flexibly changed with a rule called the d-relaxed priority rule. Here, d is a non-negative integer. For convenience, we define the problem as follows. Given a set of nodes $V = \{0, 1, \dots, n\}$ where node 0 represents the depot and the remaining nodes represent customers, an undirected graph $G = (V, E)$ is formed by V and the edge set $E = \{(i, j) : i, j \in V, i \neq j\}$ with a cost c_{ij} assigned to each $(i, j) \in E$. Customers in G are divided into g priority groups, each group V_α includes nodes with the same priority $\alpha \in P$. Here, $P = \{1, \dots, g\}$, $g \in Z^+$ is the set of priorities, sorted in descending order (i.e., g is the lowest priority represented by the highest value). We suppose a CTSP-d route and according to the d-relaxed priority rule defined in Golden et al. (2014), if z is the highest priority among the unvisited locations, the vehicle can freely visit locations with a priority $z, z + 1, \dots, z + d$ before visiting all locations in group z. This rule originates from Panchamgam (2011) and it is defined more clearly in Panchamgam et al. (2013) and Golden et al. (2014). According to Hà et al. (2022), this statement also means that if node i has a priority α then the service order of i is always smaller than that of node j with a priority $\beta > \alpha + d$, for all $\alpha, \beta \in P$.

We illustrate the d-relaxed priority rule with an application. Suppose that we have a retail company with 1 depot and 8 agents divided into 3 priority groups according to their stock levels. In Fig. 1, we present 3 solutions corresponding to 3 values of d. The priority is labeled in a circle and the depot D is labeled in a square. For convenience, we illustrate the depot with two copies. The case $d = 0$ should be used when the company wants to provide its service with a strict order of priority. However, this service often incurs high costs if agents are located far away from the depot or scattered throughout the city. If the requests are not critical, the company can increase d to reduce costs. For example, with $d = 1$, each priority level is relaxed by one, i.e., priority 1 can be served before or after priority 2, priority 2 can be served before or after priority 3, and so on, if the company has more than 3 priorities. It is easy to see that at any position with priority α, the vehicle now has more options for the next service. That is, it can move to an agent with priority $\alpha - 1$, α, or $\alpha + 1$. As a result, the vehicle is more likely to reduce costs than with $d = 0$. Finally, the case $d = 2$ allows even a higher level of relaxation because the vehicle can now serve priority 1 before or after priorities 2 and 3. In other words, the priority in this case is meaningless because the agents can be served in any order. In reality, when the stock levels of the agents are not very different or critical, one can disable the priority classification by setting the highest relaxation level.

From the application above, the d-relaxed priority rule provides a trade-off between the order of service and the service cost. Indeed, the solution cost decreases when d increases, but we have to accept that some high priority nodes may be served after lower ones. The CTSP-d has some special features depending on the d value: 1) If $d = 0$, the CTSP-d becomes the Clustered Traveling Salesman Problem with a Prespecified Order on the Clusters (CTSP-PO), as in the study of Potvin and Guertin

(1998). This problem is also known as the Ordered Cluster TSP studied by Anily et al. (1999), Ahmed (2014), and Alsheddy (2018). In the CTSP-PO, clusters must be served in a prespecified order, but the order between nodes within a cluster is not obligatory. Another problem related to the CTSP-*d* is the Traveling Salesman Problem with Hierarchical Objective Function (TSPHO) introduced by Dam et al. (2019). The TSPHO aims to minimize the total completion time of each group depending on priority order, i.e., priority 1 will be optimized first, then priority 2 and so on; 2) If $d = g - 1$, the CTSP-*d* is completely relaxed and becomes the classical TSP as explained in the example above; Thereby, considering $d > g - 1$ is not necessary; 3) a value $d \in \{1, \ldots, g-2\}$ $\forall g \geq 3$ is an intermediate case. When increasing d, we can save cost, but we also sacrifice the quality of service for high priority nodes.

Other applications can be found in Panchamgam (2011) and Panchamgam et al. (2013) in the context of the last mile distribution for humanitarian relief. Here, vulnerable people are divided into several priority groups, and satisfying the demand for relief goods such as blankets, food, and medicines is the priority objective. Hà et al. (2022) indicate some potential CTSP-*d* applications. For example, technicians in a maintenance service company can be transported to locations that require repair service or new equipment installation. The status of the air conditioning in the summer and the heat in the winter can be used to classify the priorities of customers. A similar application is proposed for internet maintenance service, where the priority of a customer is assigned based on factors such as the amount that the customer pays for the service, the urgency of the requirement, the customer's loyalty, the date of appearance on the waiting list, etc. The internet maintenance service provider chooses a value of *d* according to its service plan. After obtaining a solution, technicians will be dispatched to customers. The CTSP-*d* is also useful for situations that need a priority classification, such as using unmanned aerial vehicles (UAVs) to monitor prioritized targets or evacuating vulnerable groups in emergency situations. Considering the CTSP-*d* as a variant of the Clustered Traveling Salesman Problem (CTSP) studied by Chisman (1975), some applications are mentioned in the paper of Laporte and Palekar (2002). It should be noted that vertices in the CTSP are distributed in clusters but the priority classification is unused in this problem.

As far as we know, studies directly related to the CTSP-*d* are quite limited. The author of Panchamgam (2011) presents the first mathematical model for this problem with a simple experiment on instances with up to 30 nodes and derives several theoretical results for the worst case in Panchamgam et al. (2013). The problem is also mentioned in Golden et al. (2014). Recently, Hà et al. (2022) propose 4 formulations for the CTSP-*d* and they are the first to solve the problem on instances of up to 200 nodes with CPLEX and a metaheuristic called GILS-RVND. This metaheuristic is the combination of Iterated Local Search (ILS), Random Variable Neighborhood Descent (RVND) and Greedy Randomized Adaptive Search Procedure (GRASP). In the experiments, the authors use two types of instances, namely random and clustered, and prove that the new formulations can solve this problem more effectively than the previous model in Panchamgam (2011). More recently, Doan et al. (2021) present a problem called the Vehicle Routing Problem with Relaxed Priority Rule (VRP-RPR). This problem is developed based on the study of Panchamgam (2011). In the VRP-RPR, the heterogeneous fleet is limited by vehicle capacity, route length, and fleet size while in addition to the d-relaxed priority rule introduced above, another rule called the Order of Demand Fulfillment is applied to ensure that at the end of day, all higher priorities are served before lower ones. The VRP-RPR can be reduced to the CTSP-d when using only one vehicle with unlimited capacity and non-restricted route length. Doan et al. (2021) test the CTSP-d as a special case of the VRP-RPR with an adapted version of the Adaptive Large Neighborhood Search (ALNS) introduced by Ropke and Pisinger (2006). ALNS results surpass the GILS-RVND in all comparative criteria, especially in the aspect of runtime.

The contributions of this paper are as follows. We introduce two new Mixed Integer Linear Programming (MILP) models for the CTSP-d and an ILS-based metaheuristic to solve the problem. Results obtained on the benchmark instances of Hà et al. (2022) show that the two models can solve the problem instances faster and more efficiently than previous models in the literature. Our ILS integrates a technique to handle the d-relaxed priority rule in the operations of insertion, local search (LS), and perturbation. With powerful perturbing operators (perturbators for short) and LS operators (2-Opt, 2k-Opt, and Or-Opt), the ILS outperforms the two metaheuristics applied in the previous studies, including the GILS-RVND in Hà et al. (2022) and the ALNS in Doan et al. (2021) in terms of running time, average solution, best solution, and stability.

The remainder of the paper is structured as follows. In Sect. 2, we present two new MILP models for the CTSP-d. The ILS-based metaheuristic is described in Sect. 3 while computational experiments and results are reported in Sect. 4. Finally, we provide conclusions in Sect. 5.

2 Mathematical formulations

In this section, we first review the most effective model for the CTSP-d based on the study of Hà et al. (2022). The authors named this model F_2'. It is proven to be more effective than the original Panchamgam (2011) model. The authors used two types of variables: (i) variables u_i representing the service order of node i in the route and (ii) binary variables x_{ij} equal to 1 if the vehicle travels from i to j, and 0 otherwise.

$$F_2' : \text{Minimize} \sum_{i,j \in V} c_{ij} x_{ij} \tag{1}$$

Subject to:

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \tag{3}$$

$$u_0 = 0 \tag{4}$$

$$u_i + 1 - M_1(1 - x_{ij}) \le u_j \quad \forall (i, j) \in E, j \neq 0 \tag{5}$$

$$u_i + 1 \leq u_j \quad \forall i \in V_\alpha,\ j \in V_\beta; \forall \alpha, \beta \in \{1, \ldots, g\} : \beta > \alpha + d \tag{6}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E \tag{7}$$

$$u_i \in R^+ \quad \forall i \in V \tag{8}$$

The objective function (1) aims to minimize the total traveling cost. Constraints (2) and (3) require that the vehicle must enter and leave each node exactly once. Constraint (4) initializes the service order of the depot. Constraints (5) represent the relationship between variables u and x where M_1 is a very large number. The d-relaxed priority rule is represented by constraints (6). Finally, constraints (7) and (8) define the decision variables.

In Hà et al. (2022), the authors also use the following set of constraints to improve the model's performance:

$$\sum_{i \in V_\alpha} \sum_{j \in V_\beta} x_{ji} = 0 \qquad \forall \alpha, \beta \in P : \beta > \alpha + d \tag{9}$$

$$\sum_{i \in V_\alpha} x_{0i} = 0 \qquad \forall \alpha \in P : \alpha > 1 + d \tag{10}$$

$$\sum_{i \in V_\alpha} x_{i0} = 0 \qquad \forall \alpha \in P : \alpha < g - d \tag{11}$$

$$\sum_{i \in V_\alpha} \sum_{j \in V_\beta} x_{ij} \leq 1 \qquad \forall \alpha, \beta \in P : \beta > \alpha + d \tag{12}$$

The constraints from (9) to (12) are redundant but they allow to reduce unnecessary computational time. As mentioned above, the d-relaxed priority rule is modelled through constraints (6) with the statement that the service order u_j of a node j with priority $\beta > \alpha + d$ is always greater than service order u_i of a node i with priority α. As a result, constraints (9) are used to emphasize this rule by preventing the vehicle from traveling from j to i. Constraints (10) and (11) state that the depot is not connected with nodes that lead to violating the d-relaxed priority rule. Constraints (12) say that there is at most one trip from node i with priority α to node j with priority $\beta > \alpha + d$.

In the following, we present two new models for the CTSP-d. For node i with priority α, we denote U_i the set of nodes visited after i. Similarly, for node j with priority β, we define set U_j.

Model A

This model is based on the direct definition of the d-relaxed priority rule that if z is the highest priority among the unvisited locations, the vehicle can freely visit locations with a priority z, $z + 1$, ..., $z + d$ before visiting all locations in group with priority z. We define variables z_i to represent the highest priority among unvisited nodes starting from node i (i.e., including i itself). To avoid subtours, we adapt the single-commodity flow-based formulation in Gavish and Graves (1978) by using variables y_{ij} to indicate the flow from i to j.

Objective function (1)

Subject to:

$$
\begin{aligned}
&(2), (3), (9)-(12) && \\
&z_i \leq \alpha && \forall i \in V_\alpha && (13)\\
&z_i \geq \alpha - d && \forall i \in V_\alpha && (14)\\
&z_i - M_2(1 - x_{ij}) \leq z_j && \forall (i, j) \in E; i, j \neq 0 && (15)\\
&\sum_{j \in V} y_{ij} - \sum_{j \in V \setminus \{0\}} y_{ji} = 1 && \forall i \in V \setminus \{0\} && (16)\\
&y_{ij} \leq n.x_{ij} && \forall i \in V \setminus \{0\}, j \in V && (17)\\
&y_{ij} \geq 0 && \forall i \in V \setminus \{0\}, j \in V, i \neq j && (18)\\
&z_i \in R^+ && \forall i \in V \setminus \{0\} && (19)
\end{aligned}
$$

Constraints (13) and (14) formulate the d-relaxed priority rule to ensure the highest priority served after i is valid. Constraints (15) state that if the vehicle moves from i to j, the highest priority served from j cannot be higher than the highest priority served from i. Indeed, we have $U_j \subset U_i$ then $z_i \leq z_j$. Because variables z represent the priorities, to ensure that constraints (15) work properly, $M_2 \geq g$ must be selected. In the test we choose $M_2 = g$. Constraints (16) to (18) are flow constraints adapted from the study of Gavish and Graves (1978) to eliminate subtours. Constraints (19) indicate the possible values of z_i.

Model B

The main idea of this model is to combine variables z in model A and variables u in model F_2' into a single variable to handle the subtour elimination constraints and d-relaxed priority rule. For this purpose, each node i with priority α is now associated with a variable t_i, whose domain value is scaled up varying from $(\alpha - d)|V|$ to $(\alpha + 1)|V| - 1$. The new model is as follows:

Objective function (1)

Subject to:

$$
\begin{aligned}
&(2), (3), (9)-(12) && \\
&t_i \leq (\alpha + 1)|V| - 1 \quad \forall \alpha \in P, i \in V_\alpha && (20)\\
&t_i \geq (\alpha - d)|V| \quad \forall \alpha \in P, i \in V_\alpha && (21)\\
&t_i + 1 - M_3(1 - x_{ij}) \leq t_j \quad \forall (i, j) \in E, j \neq 0 && (22)\\
&t_i \in R^+ \quad \forall i \in V && (23)
\end{aligned}
$$

Constraints (20) and (21) show the interval of variables t_i corresponding to node i with priority α. Constraints (22) require that if the vehicle travels from i to j, t_j must be greater than t_i. In these constraints, variables t play the role of variables u in model F_2', thus allowing to eliminate the subtours. Here, M_3 is a sufficiently large number which can be estimated as $M_3 = (g + 1)|V| - 1$.

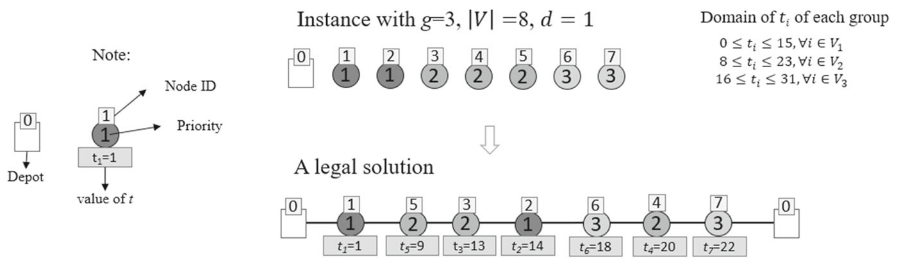
Instance with g=3, |V| =8, d = 1
Domain of t_i of each group
$0 \leq t_i \leq 15, \forall i \in V_1$
$8 \leq t_i \leq 23, \forall i \in V_2$
$16 \leq t_i \leq 31, \forall i \in V_3$
Note:
Node ID
Priority
Depot
t_1=1
value of t
A legal solution
t_1=1
t_5=9
t_3=13
t_2=14
t_6=18
t_4=20
t_7=22

Fig. 2 Illustrating how variables t_i model the d-relaxed priority rule

To illustrate more clearly the role of variables t, we show a problem instance in Fig. 2 where the graph includes 8 nodes distributed in 3 priority groups with $d = 1$. Also in this figure, a feasible route and the corresponding values of variables t are presented. We now prove that formulation B can model the d-relaxed priority rule.

Proposition 1 *Constraints (20) and (21) can model the d-relaxed priority rule.*

Proof As introduced in Sect. 1, Hà et al. (2022) prove that if a route satisfies the d-relaxed priority rule, then a node i with priority α is always served before a node j with priority $\beta > \alpha + d$. We will prove constraints (20) and (21) are equivalent to the (Hà et al. 2022) statement. Without loss of generality, we assume a node i with priority α and node j with priority $\beta > \alpha + d$. From constraints (20) and (21), we have:

$$(\alpha - d)|V| \leq t_i \leq (\alpha + 1)|V| - 1 \tag{24}$$
$$(\beta - d)|V| \leq t_j \leq (\beta + 1)|V| - 1 \tag{25}$$

Because $\beta > \alpha + d$ then $\beta - d \geq \alpha + 1$. From constraints (24) and (25), we have

$$(\alpha + 1)|V| \leq t_j \leq (\beta + 1)|V| - 1 \tag{26}$$

As a result, the combination of constraints (26) and (24) leads to $t_j > t_i$. Based on constraints (22), this ensures that node j must be served after node i in the solution, and we have the statement of Hà et al. (2022). □

3 ILS-based metaheuristic for CTSP-*d*

We now introduce our new metaheuristic for the CTSP-d. It is based on the principle of the Iterated Local Search (ILS), one efficient metaheuristic applied to successfully solve many VRP variants such as the Team orienteering problem with time windows in Vansteenwegen et al. (2009), the VRP with cross-docking in Morais et al. (2014), the Heterogeneous fleet VRP in Penna et al. (2013), etc,. Basically, the ILS starts with an initial solution ($\mathcal{S}_{init}$), then the following process is repeated until a stopping condition is reached: (1) perturb current solution ($\mathcal{S}_{curr}$) with a perturbator to generate a new solution ($\mathcal{S}_{new}$), (2) perform local search to achieve a local optimum ($\mathcal{S}^*_{new}$), and (3) check $\mathcal{S}^*_{new}$ with acceptance conditions and update the global best solution ($\mathcal{S}_{best}$). In this study, we use the ILS with different perturbators and LS operators adapted to the CTSP-d. The algorithm and its operators are shown in Algorithm 1.

Algorithm 1: ILS for CTSP-d

```
Input  : Graph, list of perturbators $\mathcal{P}$, list of local searches $\mathcal{L}$, parameters
Output: $\mathcal{S}_{best}$
1  begin
2      * Construct an initial solution $\mathcal{S}_{init}$
3      * $\mathcal{S}_{curr} \leftarrow$ improve $\mathcal{S}_{init}$ with local searches in $\mathcal{L}$
4      * $\mathcal{S}_{best} \leftarrow \mathcal{S}_{curr}$
5      for $iteration = 1$ to $w$ do
6          * Randomly select perturbator $\pi^*$ in $\mathcal{P}$
7          * $\mathcal{S}_{new} \leftarrow$ perturb $\mathcal{S}_{curr}$ with $\pi^*$
8          * $\mathcal{S}^*_{new} \leftarrow$ improve $\mathcal{S}_{new}$ with local searches in $\mathcal{L}$
9          if $f(\mathcal{S}^*_{new}) < f(\mathcal{S}_{best})$ then
10             $\mathcal{S}_{best} \leftarrow \mathcal{S}_{curr} \leftarrow \mathcal{S}^*_{new}$
11         end
12         if $f(\mathcal{S}^*_{new}) \leq \Delta.f(\mathcal{S}_{curr})$ then
13             $\mathcal{S}_{curr} \leftarrow \mathcal{S}^*_{new}$
14         end
15     end
16 end
```

In Algorithm 1, we perform ILS with w iterations. A random perturbator π^* shakes current solution $\mathcal{S}_{curr}$ to generate a new solution $\mathcal{S}_{new}$. Then, local search operators improve $\mathcal{S}_{new}$ to find a better solution $\mathcal{S}_{new^*}$. The global best solution $\mathcal{S}_{best}$ is updated if $\mathcal{S}^*_{new}$ yields a new best cost. In addition, to increase the algorithm diversity, we accept not too bad solutions generated during the search by updating $\mathcal{S}_{curr}$ with $\mathcal{S}^*_{new}$ if the objective value of $\mathcal{S}^*_{new}$ is not greater than $\Delta.f(\mathcal{S}_{curr})$, where $\Delta \geq 1$ is a given value.

3.1 Handling the *d*-relaxed priority rule

Basically, operations such as building a new route, repairing a route, LS, and perturbation, manipulate a solution with two main actions: removal and insertion. Removal commonly does not require any treatment, but insertion requires the consideration of constraints. In the CTSP-d, the d-relaxed priority rule is the main constraint that we need to consider. More specifically, constraints (6) simulating this rule show that if a node i with a priority α exists in the current route, then an unrouted node j with a priority $\beta > \alpha + d$ must be inserted after i. We use the method presented in Doan et al. (2021) to handle this rule, i.e., finding a Valid Insertion Range (VIR) for each unrouted node before inserting it into the route. We use the route structure as described in Fig. 1. From left to right, the first depot is called the starting depot ($index = 0$) and the last depot is called the ending depot ($index = n + 1$). The VIR of a node i with a priority α is defined from the starting index i_s to the ending index i_e as follows:

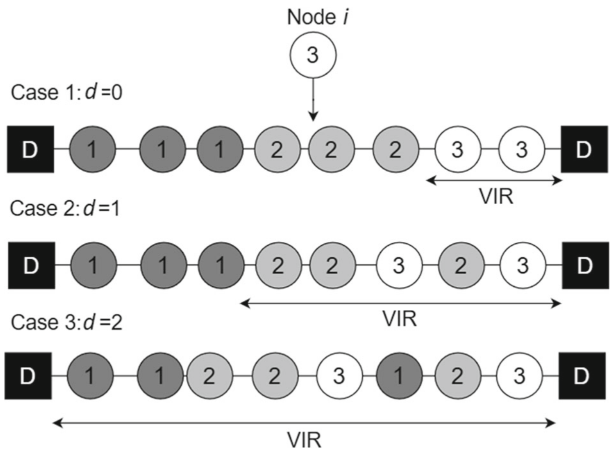

Fig. 3 Valid insertion range of node i

- *Starting index*: $i_s = i_{highest} + 1$ where $i_{highest}$ is the highest index of nodes with priorities less than $\alpha - d$. If such a node does not exist, $i_s = 1$. The rule is held because node i is always visited after a node with priority $\alpha - d - 1$ or smaller.
- *Ending index:* i_e is the lowest index of nodes with priorities greater than $\alpha + d$. If such a node does not exist, i_e is set to the index of the ending depot. The rule is held because node i is always visited before a node with priority $\alpha + d + 1$ or greater.

In Fig. 3, we illustrate how to find the VIR of a node i with priority 3 when inserting i into partial routes. We assume 3 routes corresponding to 3 cases of d. With $d = 0$, node i can only be inserted into the range after the last node of priority 2. Then, with $d = 1$, i can only be inserted into the range after the last node of priority 1. Finally, with $d = 3$, the CTSP-d is completely relaxed and i can be inserted into any position.

To find the VIR in $\mathcal{O}(1)$, we use variables to memorize the indices of the last and first nodes for each priority. Every time a node i with priority α is inserted into the route, we update the corresponding indices for α.

3.2 Initial solution

The initial solution is built on the cheapest insertion principle. We denote $E_{\mathcal{S}}$ the set of edges of a route $\mathcal{S}$ where each edge (u, v) has a cost c_{uv}. In the cheapest insertion, each unrouted node i is inserted into (u^*, v^*) such that:

$$(u^*, v^*) = \underset{(u,v)\in E_s}{\arg\min}(c_{ui} + c_{iv} - c_{uv}) \tag{27}$$

3.3 Perturbator

We introduce three perturbators in this subsection: m-relocate, m-swap, and m-segment shuffle. Here m is the number of actions performed in each ILS iteration by the corresponding perturbator. The value m of perturbator i is selected randomly in the interval $[a_i, b_i]$. The appropriate interval is defined for each perturbator in preliminary tests.

- ***m***-relocate: randomly select an integer $m \in [a_1, b_1]$ and relocate m nodes to other positions in the route. A relocation needs to consider the VIR of the node. After preliminary tests, we choose $[a_1, b_1] = [5, 12]$.
- ***m***-swap: randomly select an integer $m \in [a_2, b_2]$ and swap m pairs of nodes in the route. The swap pair (i, j) violates the d-relaxed priority rule if $\beta - \alpha > d$, where α, β are priorities of nodes i, j, respectively. After preliminary tests, we choose $[a_2, b_2] = [5, 25]$.
- ***m***-segment shuffle: randomly select an integer $m \in [a_3, b_3]$ and shuffle m segments of nodes. To perform a shuffle considering the d-relaxed priority rule, we randomly choose a priority α and detect a valid segment to shuffle. This segment has a length equal to the VIR of α. After preliminary tests, we choose $[a_3, b_3] = [1, 3]$.

It should be noted that, Hà et al. (2022) use the perturbation that is also based on the relocation and swap operations. But they do not have m-segment shuffle. In addition, to check the feasibility of the perturbation move w.r.t the d-relaxed priority rule, they use a time–costly procedure, while we can perform the same action faster with the VIR.

3.4 Local search

The local search plays the most important role in the ILS algorithm. It starts from a candidate solution and then iteratively moves to a neighbor solution to search for an improved solution. Several classic local search operators can be used, such as swapping two strings or relocating a string of nodes in the solution, as in Hà et al. (2022). Doan et al. (2021) propose 2-Opt to design the ALNS for solving the CTSP-d. In this paper, we use three local search operators: 2-Opt, Or-opt, and 2k-Opt, in which the two latter operators have never been used to tackle the CTSP-d. Note that, these local searches create neighborhoods which are larger than the ones proposed in Hà et al. (2022), but our ILS can check the feasibility of the moves more efficiently using dedicated techniques. To describe in details our local search operators, for simplicity, we denote node i be the name of the node at position i in a feasible route. The operators work as follows:

2-Opt: Assuming that we have two positions i and j ($j \geq i + 3$) on a route, a 2-Opt move reverses path s from $i + 1$ to $j - 1$ and replaces arcs $(i, i + 1)$ and $(j - 1, j)$ by arcs $(i, j - 1)$ and $(i + 1, j)$. This results in a cost reduction calculated by $\Delta_{2Opt}(i, j) = c_{(i,i+1)} + c_{(j-1,i)} - c_{(i,j-1)} - c_{(i+1,j)}$. Here, $c_{(i,i+1)}$ is the cost of arc $(i, i + 1)$ and similarly for the other costs. To verify if a 2-Opt move is feasible w.r.t. the d-relaxed rule, we need to ensure that on path s there does not exist any pair of nodes with priorities α and β such that $|\beta - \alpha| > d$. Indeed, if we can find such a pair of nodes on path s, according to constraint (6), the node of priority α must be served before the node of priority β. A 2-Opt move results in reversing s and also reverses the service order between node of priority α and node of priority β. This move is infeasible because constraint (6) is violated. On the other hand, if the aforementioned pair of nodes does not exist in s, for any pair of nodes with priorities α and β, constraint (6) holds for the reversal path. Based on this characteristic, we can quickly detect a feasible 2-Opt move if the difference between the maximum and minimum priorities of nodes on s is not greater than d.

Or-Opt: This operator is a restricted case of 3-Opt. An Or-Opt move shifts a segment of 1, 2 or 3 consecutive nodes to another position in order to achieve a better route cost. To perform an Or-Opt move adapted to the d-relaxed priority rule, we find the first edge such that: (1) this edge is a common edge in the VIRs of all nodes in the moving segment, and (2) moving the segment to this edge yields a reduction in the route cost.

2k-Opt: Since the sequence reversal in 2-Opt often violates the d-relaxed priority rule, we introduce 2k-Opt operator, which includes multiple nested 2-Opt moves, to overcome this issue. The 2k-Opt is a new local search presented recently by Pacheco et al. (2022) for solving the Pickup-and-Delivery Traveling Salesman Problem (PDTSP). It is based on dynamic programming and adapted for our problem as follows.

In a CTSP-d route, let $F(i, j)$ denote the cost of the best 2-Opt combination in path $(i, \ldots, j)$ and $R(i, j)$ denote the cost of the best 2-Opt combination in reversal path $(j, \ldots, i)$. $\Delta_{2Opt}(i, j)$ is the cost reduction when applying a 2-Opt move in $(i, \ldots, j)$ as we stated in the 2-Opt operator above where nodes i and j are assigned priorities α and β, respectively. Finally, in the reversal path $(j, \ldots, i)$, we assume a pair (u, v) (u is served before v) with priorities p^{max} and p^{min}, that are the maximum and minimum values of priorities in $(j, \ldots, i)$, respectively.

$F(i, j)$ can be evaluated by induction, selecting the best out of three cases:

- Searching for the best 2-Opt combination on path $(i + 1, \ldots, j)$;
- Searching for the best 2-Opt combination on path $(i, \ldots, j - 1)$;
- Applying the move and searching for additional moves in the reversed sequence on path $(j - 1, \ldots, i + 1)$.

The following equations represent the policy above:

$$F(i, j) = \begin{cases} 0 & \text{if } j \leq i + 1 \\ \min \begin{cases} F(i+1, j) \\ F(i, j-1) \\ \Delta_{2Opt}(i, j) + R(i+1, j-1) \end{cases} & \text{otherwise} \end{cases} \tag{28}$$

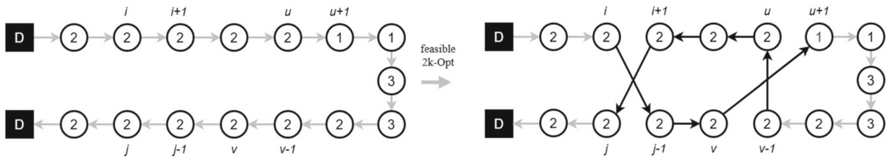

Fig. 4 An example of 2k-Opt move for the CTSP-d ($d = 1$)

Similarly to $F(i, j)$, evaluating $R(i, j)$ can be done as follows.

$$R(i,j) = \begin{cases} \infty & \text{if } p^{max} - p^{min} > d \\ 0 & \text{if } j = i \\ 0 & \text{if } j = i+1 \text{ and } \beta - \alpha \le d \\ \infty & \text{if } j = i+1 \text{ and } \beta - \alpha > d \\ \min \begin{cases} R(i+1, j) \\ R(i, j-1) \\ \Delta_{2Opt}(i,j) + F(i+1, j-1) \end{cases} & \text{otherwise} \end{cases} \tag{29}$$

Figure 4 illustrates an example with a feasible 2k-Opt move including two nested 2-Opt moves in a CTSP-d route with $d = 1$. Numbers in circles represent priorities and letter D in square represents the depot. The outer 2-Opt reverses path $(i+1, \ldots, j-1)$ and replaces arcs $(i, i+1)$ and $(j-1, j)$ with arcs $(i, j-1)$ and $(i+1, j)$. The inner 2-Opt reverses path $(u+1, \ldots, v-1)$ and replaces arcs $(u, u+1)$ and $(v-1, v)$ with arcs $(v-1, u)$ and $(v, u+1)$. The new links are marked in bold. To process the move, which includes two nested 2-Opt moves, we only need to reverse the paths $(i+1, \ldots, u)$ and $(v, j-1)$. After performing the 2k-Opt move, the route remains feasible w.r.t the d-relaxed priority rule because with $d = 1$, nodes of priority 3 have to be served after those of priority 1 but nodes of both priorities 1 and 3 can be served before or after priority-2 nodes. If we perform 2-Opt move for this route, both the outer 2-Opt and inner 2-Opt moves are infeasible because it requires to reverse the path from $u+1$ to $v-1$, which leads to priority-3 nodes being served before priority-1 nodes. Despite of the exponential-size neighborhoods, the 2k-Opt moves can be done in $\mathcal{O}(n^2)$, which is equivalent to other classical local searches.

4 Computational experiments

In this section, we first present the experiment setup and the benchmark instances in Sect. 4.1. In Sect. 4.2, we provide information about parameters used in ILS and analyze the impact of each LS operator on the performance of the algorithm. We compare the CTSP-d models in Sect. 4.3.1. Finally, the comparison between metaheuristics is carried out in Sect. 4.3.2.

4.1 Instance and experiment setup

We use the instance set of Hà et al. (2022), which can be downloaded at http://orlab.vn/home/download to test our models and metaheuristic. The set includes instances with the number of nodes varying from 42 to 200 nodes. Depending on the distribution of nodes, two types of instance are created: type R (random) and type C (clustered). Type R represents typical commercial situations where customers are randomly distributed and their priority depends on their storage level. Type C represents disaster situations such as earthquakes or tsunamis, where people are gathered into clusters and the priority depends on customer urgency. Hà et al. (2022) name the instances in the format X-Y-g-d, where X is the name of original TSPLIB instances, Y represents the instance type. The notations g and d are self-defined. For example, the instance *KroA100-R-3-0* is generated from the instance KroA100 of the TSPLIB of Reinelt (1991) with 100 nodes, and it belongs to type R with 3 priority groups and the value of d is zero. In addition, for each instance generated from Swiss42 and Berlin52, three copies denoted with a letter a, b, c at the end of the name have randomly been generated. Based on the benchmark instances from the literature, we use 3 sets for our experiments as follows:

1. Set 1: The instances in this set are used for parameter tuning and LS sensitivity analysis in Sect. 4.2. As reported in Hà et al. (2022), type-R instances is harder than type-C ones for a reasonable computational effort, this set contains all type-R instances with 5 priority groups and instance sizes of 100 and 200 nodes.
2. Set 2: The instances in this set are used to investigate the performance of the models in Sect. 4.3.1. This set includes all the small instances with prefixes *Swiss42* and *Berlin52* created by Hà et al. (2022). In addition, to observe more clearly the behavior of the MILP formulations, we add in this set 24 larger instances with 60 and 70 nodes created from the TSPLIB instances *kroA100-R-5*, *kroB100-R-5*, *kroA100-C-5*, and *kroB100-C-5*. To generate a new instance, we choose random nodes from the original instance until achieving the desired instance size. In total, we have 88 instances divided equally for both types.
3. Set 3: This set includes all the instances proposed by Hà et al. (2022) with the size from 52 to 200 nodes. They are used in Sect. 4.3.2 to investigate the efficiency of our metaheuristic.

Our experiments are performed on a computer with Intel core(TM) i3-6100 3.7GHz CPU, 16GB DDR3 and Microsoft Windows 10 OS. The ILS algorithm is programmed with Java 10, and the MILP models are solved with CPLEX 12.9 using OPL modeling language. It is worth noting that the model F_2' is re-implemented and run on our machine while the results of the reference metaheuristics ALNS and GILS-RVND are taken from Doan et al. (2021) and Hà et al. (2022).

4.2 Parameters and LS analysis

In this subsection, we describe how to tune the ILS parameters and analyse the importance of LS operators, the main component of our metaheuristic. The experiment is performed on the first set of instances. The results of the ILS algorithm for each

Table 1 Sensitivity analysis for LS operators

Setting	Average cost (%)	Best cost (%)	Runtime (s)
Full option	0.00	0.00	40.52
Without 2k-Opt	1.25	0.49	63.65
Without 2-Opt	1.19	0.47	50.73
Without Or-Opt	1.66	1.12	27.85

instance are collected after 5 runs, and the average cost, best cost, and average running time in seconds are reported.

For the total number of iterations w, we test four values $\{3000, 5000, 10{,}000, 20{,}000\}$ and observe that at $w = 3000$ and $w = 10{,}000$, average runtimes of ILS are approximately equivalent to those values for ALNS and GILS-RVND, respectively. It should be mentioned that we use the same computer as (Doan et al. 2021) while (Hà et al. 2022) use a computer with a CPU, according to Doan et al. (2021), about 83.2% faster than ours. In addition, if we set the value of w greater than 10,000, the running time of the algorithm increases proportionally, but the obtained solutions are not improved significantly. Thus, we decide to use two settings $w = 3000$ and $w = 10{,}000$ in the following experiments. Parameter Δ is tuned at $w = 3000$ with 7 values $\{1, 1.01, 1.02, 1.03, 1.04, 1.05, \infty\}$. After the tuning phase, we choose $\Delta = 1.02$ for instances with up to 100 nodes and $\Delta = 1.05$ for instances with 200 nodes.

We observe the impact of LS operators through a sensitivity analysis with $w = 3000$ and the tuned Δ values mentioned above. We remove each LS operator one by one and compare the obtained results with the results provided by the full option setting of the algorithm. As can be seen in Table 1, the positive values in Columns "Average cost" and "Best cost" show that all the LS operators have contribution to the overall performance of the algorithm. In more details, Or-Opt has the strongest impact and is also the most time-consuming operator. Two operators 2k-Opt and 2-Opt have approximately similar impact, but 2k-Opt seems to be better than 2-Opt when being coupled with Or-Opt.

4.3 Experiment results

4.3.1 MILP model comparison

We now demonstrate the performance of the new models in comparison with model F_2'. The small instances in Set 2 are selected for the experiment. The running time for each instance is limited to 3600 s. In Tables 2 and 3, Columns “Instance", “Sol”, “Gap”, and “Time” report instance name, objective value, gaps returned by CPLEX, and the running time in seconds, respectively. For two instances *kroA70-R-5-1* and *kroB70-R-5-1* executed with model F_2', we put “-" in solution value and gap to imply that CPLEX cannot find any solution. As a result, we cannot show the average result of all instances executed with this model.

On type-R instances, when comparing available results, we see that the average solutions and gaps of models A and B are often better than those of model F_2'. Models A and B are also faster than model F_2' in terms of average running time. When compared to model B, model A is about 1.5% faster. Regarding the number of proved optimal solutions, models A and B both find 26 optimal solutions, while this number of model F_2' is 25. On type-C instances, we find more optimal solutions compared to type-R instances. This is similar to the observation in Hà et al. (2022) that the clustered instances are easier than the random ones for solution methods. Model A dominates the other formulations on type-C instances. It finds 42 optimal solutions, while model B and model F_2' both solve 36 instances to optimality. Model A requires the smallest computing time. We believe that the reason for this phenomena is due to the efficiency of the single-commodity flow when dealing with subtour elimination. If we compare model B and F_2', model B achieves better average cost and gap than model F_2'. However, the average runnning time of model B is longer than that of model F_2'.

In overall, our new models work better than F_2'. On both instance types, models A and B achieve better average results and gaps. Model A tends to work more efficiently on C-type instances while Model B performs better on R-type instances. We also observe that $d = 0$ is often the easiest case because most instances are solved to optimality, regardless of instance type. Based on the characteristics of the CTSP-d, we can explain that, when d is set to zero, the performance of the MILP models is increased because customers have to be served in the strict order of priority. When we increase d (i.e., relaxing priorities), the customers with priority p have more choices to be served before or after customers with priorities from p to $p + d$. Therefore, the search space is expanded when we set $d = \{1, 3\}$.

4.3.2 Metaheuristic comparison

In this subsection, we discuss the results of our ILS compared to two metaheuristics of previous studies, including GILS-RVND in Hà et al. (2022) and ALNS in Doan et al. (2021). ILS is executed with two versions: fast with 3000 iterations (ILS-3000) and slow with 10,000 iterations (ILS-10000). On each instance, the metaheuristics are run 10 times and the average solutions, best solutions, and running times on average are recorded for the comparison.

In Table 4, we summarize the comparison results. The three last column headers show the criteria to compare while the row headers show each pair of compared algorithms. Here, the values in a cell are in format X-Y-Z representing the number of instances on which a method performs better (X), equally (Y), or worse (Z) than another. For example, values 80-34-2 located in row "ILS-3000 vs GILS-RVND" and column "Average cost", mean that: compared to GILS-RVND in terms of average cost, ILS-3000 is better on 80 instances, equal on 34 instances, and worse on 2 instances.

As can be observed in Table 4, both ILS settings outperform GILS-RVND and ALNS on all criteria. More remarkably, the two ILS settings can find 13 new best known solutions. Most of them come from instances with 200 nodes. To better observe the speed of the different metaheuristics, we illustrate the average running time per run of the algorithms on type-R instances in Fig. 5. The running times on type-C instances have the same pattern.

Table 2 Comparison between new models and model F_2' on instances of type R

Instance	Model A			Model B			F_2'		
	Sol	Time	Gap	Sol	Time	Gap	Sol	Time	Gap
swiss42-R-1-0-a	1273	1.20	0.00	1273	1.50	0.00	1273	1.14	0.00
swiss42-R-3-0-a	2256	0.80	0.00	2256	0.55	0.00	2256	1.13	0.00
swiss42-R-3-0-b	2153	2.42	0.00	2153	0.86	0.00	2153	2.69	0.00
swiss42-R-3-0-c	2080	1.55	0.00	2080	4.14	0.00	2080	9.64	0.00
swiss42-R-3-1-a	1652	175.50	0.00	1652	108.92	0.00	1652	234.11	0.00
swiss42-R-3-1-b	1607	362.86	0.00	1607	346.36	0.00	1607	908.06	0.00
swiss42-R-3-1-c	1524	406.56	0.00	1525	1324.59	0.00	1525	1193.25	0.00
swiss42-R-5-0-a	2365	0.70	0.00	2365	0.69	0.00	2365	2.77	0.00
swiss42-R-5-0-b	2567	0.69	0.00	2567	0.48	0.00	2567	2.09	0.00
swiss42-R-5-0-c	2694	0.86	0.00	2694	0.48	0.00	2694	1.34	0.00
swiss42-R-5-1-a	1812	193.66	0.00	1812	127.88	0.00	1812	528.45	0.00
swiss42-R-5-1-b	1905	3370.61	0.00	1905	2817.30	0.00	1906	3600.00	8.90
swiss42-R-5-1-c	1910	1347.88	0.00	1910	866.38	0.00	1910	3452.56	0.00
swiss42-R-5-3-a	1474	3600.00	2.88	1511	3600.00	5.88	1474	3600.00	2.49
swiss42-R-5-3-b	1594	3600.00	10.35	1560	3600.00	7.71	1560	3600.00	7.27
swiss42-R-5-3-c	1513	3600.00	1.79	1510	1565.30	0.00	1510	2773.48	0.00
berlin52-R-1-0-a	7542	1.50	0.00	7542	2.45	0.00	7542	2.78	0.00
berlin52-R-3-0-a	12,765	1.63	0.00	12,765	6.98	0.00	12,765	11.19	0.00
berlin52-R-3-0-b	12,668	3.28	0.00	12,668	2.91	0.00	12,668	5.38	0.00
berlin52-R-3-0-c	12,483	3.91	0.00	12,483	6.20	0.00	12,483	15.47	0.00
berlin52-R-3-1-a	9473	1323.78	0.00	9473	3441.77	0.00	9473	3359.40	0.00
berlin52-R-3-1-b	9419	3600.00	1.49	9442	3600.00	2.64	9511	3600.00	4.06
berlin52-R-3-1-c	9577	3600.00	1.75	9577	3600.00	3.31	9744	3600.00	5.78
berlin52-R-5-0-a	16,414	1.22	0.00	16,414	0.84	0.00	16,414	3.52	0.00
berlin52-R-5-0-b	13,759	2.13	0.00	13,759	11.72	0.00	13,759	28.08	0.00
berlin52-R-5-0-c	14,131	1.31	0.00	14,131	4.30	0.00	14,131	7.05	0.00
berlin52-R-5-1-a	11,662	3600.00	7.40	11,742	3600.00	8.18	13,321	3600.00	20.94
berlin52-R-5-1-b	10,022	3600.00	5.44	10,069	3600.00	5.52	10,069	3600.00	8.20
berlin52-R-5-1-c	10,940	3600.00	8.55	11,020	3600.00	8.90	11,038	3600.00	10.02
berlin52-R-5-3-a	9079	3600.00	7.18	9078	3600.00	7.69	9085	3600.00	8.00
berlin52-R-5-3-b	8036	2399.45	0.00	8036	3600.00	1.08	8036	3600.00	0.63
berlin52-R-5-3-c	8224	3600.00	2.68	8224	3600.00	2.47	8224	3600.00	2.30
kroA60-R-5-0	38,973	4.75	0.00	38,973	36.76	0.00	38,973	141.23	0.00
kroA70-R-5-0	40,854	18.31	0.00	40,854	43.47	0.00	40,854	90.77	0.00
kroB60-R-5-0	39,705	2.14	0.00	39,705	6.34	0.00	39,705	20.84	0.00

Table 2 continued

Instance	Model A			Model B			F_2'		
	Sol	Time	Gap	Sol	Time	Gap	Sol	Time	Gap
kroB70-R-5-0	42,489	23.47	0.00	42,489	55.97	0.00	42,489	106.13	0.00
kroA60-R-5-1	29,222	3600.00	15.13	28,712	3600.00	12.22	35,914	3600.00	34.01
kroA70-R-5-1	30,062	3600.00	13.68	29,870	3600.00	11.79	–	3600.00	–
kroB60-R-5-1	28,926	3600.00	10.56	28,817	3600.00	9.81	30,310	3600.00	16.45
kroB70-R-5-1	33,588	3600.00	16.99	32,380	3600.00	13.01	–	3600.00	–
kroA60-R-5-3	21,736	3600.00	17.04	20,852	3600.00	13.95	22,448	3600.00	20.93
kroA70-R-5-3	24,844	3600.00	24.28	25,385	3600.00	27.00	26,515	3600.00	29.89
kroB60-R-5-3	21,678	3600.00	16.06	21,589	3600.00	16.29	23,962	3600.00	24.69
kroB70-R-5-3	26,630	3600.00	21.27	26,306	3600.00	21.22	30,333	3600.00	31.76
Average	13301.8	1692.09	4.19	13244.0	1717.84	4.06	–	1847.79	–

Table 3 Comparison between new models and model F_2' on instances of type C

Instance	Model A			Model B			F_2'		
	Sol	Time	Gap	Sol	Time	Gap	Sol	Time	Gap
swiss42-C-1-0-a	1273	1.11	0.00	1273	1.52	0.00	1273	1.27	0.00
swiss42-C-3-0-a	1347	1.44	0.00	1347	5.91	0.00	1347	3.74	0.00
swiss42-C-3-0-b	1505	2.27	0.00	1505	17.58	0.00	1505	13.83	0.00
swiss42-C-3-0-c	1467	1.06	0.00	1467	1.45	0.00	1467	3.63	0.00
swiss42-C-3-1-a	1301	4.72	0.00	1301	11.22	0.00	1301	6.33	0.00
swiss42-C-3-1-b	1344	8.47	0.00	1344	26.66	0.00	1344	12.48	0.00
swiss42-C-3-1-c	1357	4.78	0.00	1357	11.66	0.00	1357	3.19	0.00
swiss42-C-5-0-a	1561	1.89	0.00	1561	382.69	0.00	1561	281.80	0.00
swiss42-C-5-0-b	1540	1.13	0.00	1540	2.73	0.00	1540	4.74	0.00
swiss42-C-5-0-c	1532	1.45	0.00	1532	29.53	0.00	1532	77.44	0.00
swiss42-C-5-1-a	1434	6.72	0.00	1434	18.39	0.00	1434	35.49	0.00
swiss42-C-5-1-b	1469	15.22	0.00	1469	39.77	0.00	1469	84.31	0.00
swiss42-C-5-1-c	1334	1.55	0.00	1334	2.38	0.00	1334	4.69	0.00
swiss42-C-5-3-a	1273	1.97	0.00	1273	1.64	0.00	1273	1.30	0.00
swiss42-C-5-3-b	1273	2.23	0.00	1273	3.11	0.00	1273	3.31	0.00
swiss42-C-5-3-c	1301	3.70	0.00	1301	9.83	0.00	1301	9.17	0.00
berlin52-C-1-0-a	7542	1.81	0.00	7542	3.03	0.00	7542	3.76	0.00
berlin52-C-3-0-a	8144	1.38	0.00	8144	3.08	0.00	8144	3.98	0.00
berlin52-C-3-0-b	8016	1.08	0.00	8016	1.67	0.00	8016	4.38	0.00
berlin52-C-3-0-c	9085	2.05	0.00	9085	884.66	0.00	9085	2031.02	0.00
berlin52-C-3-1-a	7952	7.98	0.00	7952	12.53	0.00	7952	11.39	0.00
berlin52-C-3-1-b	7596	1.95	0.00	7596	11.91	0.00	7596	8.92	0.00
berlin52-C-3-1-c	7984	3.66	0.00	7984	2.42	0.00	7984	3.63	0.00

Table 3 continued

Instance	Model A			Model B			F_2'		
	Sol	Time	Gap	Sol	Time	Gap	Sol	Time	Gap
berlin52-C-5-0-a	9430	2.14	0.00	9430	5861.00	0.00	9430	72.34	0.00
berlin52-C-5-0-b	8669	1.51	0.00	8669	8.59	0.00	8669	17.03	0.00
berlin52-C-5-0-c	9651	8.55	0.00	9651	3600.00	5.98	9651	3600.00	6.01
berlin52-C-5-1-a	8811	2.72	0.00	8811	8.14	0.00	8811	12.06	0.00
berlin52-C-5-1-b	7948	5.49	0.00	7948	1.47	0.00	7948	2.98	0.00
berlin52-C-5-1-c	8509	4.44	0.00	8509	17.05	0.00	8509	21.22	0.00
berlin52-C-5-3-a	7972	3600.00	1.65	7907	3600.00	1.96	7907	3600.00	1.13
berlin52-C-5-3-b	7614	3.72	0.00	7614	1.11	0.00	7614	2.42	0.00
berlin52-C-5-3-c	7631	5.36	0.00	7631	4.48	0.00	7631	5.66	0.00
kroA60-C-5-0	20,372	8.48	0.00	20,372	3600.00	2.20	20,372	3600.00	2.78
kroA70-C-5-0	22,218	7.19	0.00	22,218	3600.00	4.71	22,218	3600.00	4.84
kroB60-C-5-0	20,937	2.59	0.00	20,937	24.33	0.00	20,937	27.88	0.00
kroB70-C-5-0	22,501	5.47	0.00	22,501	22.99	0.00	22,501	81.72	0.00
kroA60-C-5-1	18,593	355.20	0.00	18,593	3600.00	2.48	18,903	3600.00	4.26
kroA70-C-5-1	20,577	3600.00	0.70	20,577	3600.00	4.66	20,577	3600.00	4.81
kroB60-C-5-1	19,000	211.64	0.00	19,000	3600.00	3.89	19,000	3600.00	3.29
kroB70-C-5-1	20,848	268.31	0.00	20,913	3600.00	4.44	20,956	3600.00	5.74
kroA60-C-5-3	17,399	42.76	0.00	17,399	438.08	0.00	17,399	418.34	0.00
kroA70-C-5-3	19,274	40.17	0.00	19,274	1134.44	0.00	19,274	896.06	0.00
kroB60-C-5-3	17,210	22.23	0.00	17,210	578.91	0.00	17,210	466.77	0.00
kroB70-C-5-3	19,197	72.19	0.00	19,197	2712.75	0.00	19,197	2849.33	0.00
Average	8931.6	189.77	0.05	8931.6	934.06	0.69	8939.6	824.72	0.75

Table 4 Efficiency comparison between metaheuristics

Comparison pair	Average cost	Best cost	Runtime
ILS-3000 vs GILS-RVND	80-34-2	22-92-2	116-0-0
ILS-10000 vs GILS-RVRD	82-34-0	23-93-0	78-0-38
ILS-3000 vs ALNS	47-55-14	11-102-3	99-0-17
ILS-10000 vs ALNS	53-57-6	13-103-0	0-0-116
ILS-10000 vs ILS-3000	28-84-4	6-110-0	0-0-116

The detailed results for each instance are presented in Tables 5 and 6 via 3 wide columns: “Best cost”, “Gap (%)", and “Runtime (s)". Here, values in Column “Gap (%)” represent the difference in percentage between the average cost (aC) and the best cost (bC) over 10 runs, which is calculated according to the formula: $Gap = (aC - bC).100/bC$. The “Average” row shows the average values of all instances in the corresponding table.

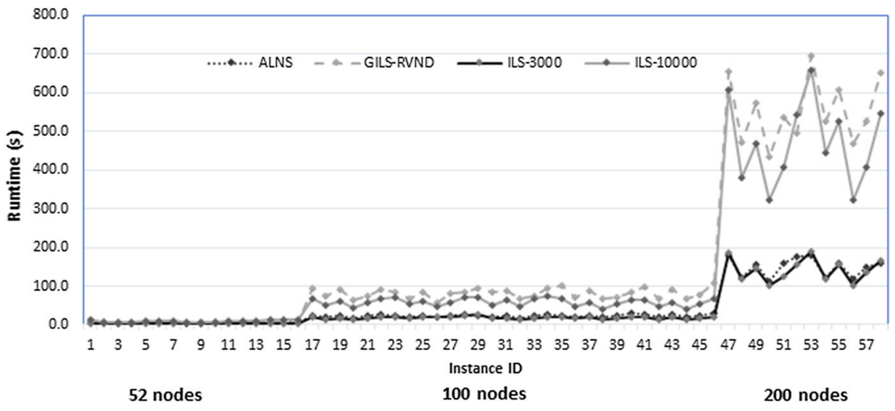

Fig. 5 Average running time per run of the metaheuristics tested on type-R instances

The results show that both ILS versions provide more stable results. In the "Average" row, the average gap of ILS methods is below 0.1% on both instance types. Especially, on type-C instances, average gaps of both ILS-3000 and ILS-10000 are less than 0.01%. These results also show that solutions achieved on type C tend to be more stable than those on type R. With the fast setting (i.e., ILS-3000), ILS already outperforms the other algorithms in all comparison criteria. Using the slow setting (i.e., ILS-10000), the algorithm is more stable and we find 6 additional best known solutions. Although the average runtime of ILS-10000 is higher than that of ALNS, but it is about 18.1% lower than that of GILS-RVND.

5 Conclusion

In this paper, we have studied the CTSP-*d*, an optimization problem with multiple applications, especially in situations in which customers need to be classified into several priority groups. The main feature of the problem is the capability to control service plans with a rule called the *d*-relaxed priority rule. By changing the value of *d* in the rule, we can make service plans ranging from strict order of priorities to the classic TSP without any order requirement. We have proposed two new MILP models that perform better than previous models. We also have developed an ILS-based metaheuristic with operators adapted to our problem. The results obtained on the instance set of Hà et al. (2022) show that our ILS with two settings, namely ILS-3000 and ILS-10000, outperforms ALNS in Doan et al. (2021) and GILS-RVND in Hà et al. (2022) on all comparison criteria such as stability, solution quality, and running time. In total, 13 best known solutions have been improved in this research.

For future research, it could be interesting to develop exact methods based on our new models to solve the largest CTSP-*d* instances. To achieve this goal, new valid inequalities could be proposed to design efficient branch-and-cut algorithms. Solving the problem with population-based metaheuristics (e.g,. genetic algorithm)

Table 5 Metaheuristic comparison on the instances of type R

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
berlin52R-1-0-a	7542	7542	7542	7542	0.00	0.00	0.00	0.00	7.9	7.7	3.1	12.1
berlin52R-3-0-a	12,765	12,765	12,765	12,765	0.00	0.00	0.00	0.00	5.2	6.3	2.1	8.4
berlin52R-3-0-b	12,668	12,668	12,668	12,668	0.00	0.00	0.00	0.00	5.0	6.5	2.0	8.0
berlin52R-3-0-c	12,483	12,483	12,483	12,483	0.00	0.00	0.00	0.00	5.0	6.4	2.0	8.1
berlin52R-3-1-a	9473	9473	9473	9473	0.00	0.00	0.00	0.00	6.0	8.3	3.1	11.2
berlin52R-3-1-b	9419	9419	9419	9419	0.00	0.00	0.00	0.00	6.2	7.0	3.0	10.4
berlin52R-3-1-c	9577	9577	9577	9577	0.00	0.00	0.00	0.00	6.0	8.3	3.0	10.0
berlin52R-5-0-a	16,414	16,414	16,414	16,414	0.00	0.00	0.00	0.00	5.0	5.7	2.0	7.0
berlin52R-5-0-b	13,759	13,759	13,759	13,759	0.00	0.00	0.00	0.00	5.0	5.8	2.0	7.5
berlin52R-5-0-c	14,131	14,131	14,131	14,131	0.00	0.00	0.00	0.00	5.1	6.5	2.0	7.4
berlin52R-5-1-a	11,651	11,651	11,651	11,651	0.46	0.00	0.00	0.00	6.3	6.9	3.0	10.1
berlin52R-5-1-b	9957	9957	9957	9957	0.07	0.07	0.00	0.00	6.0	6.2	3.0	10.3
berlin52R-5-1-c	10,940	10,940	10,940	10,940	0.00	0.00	0.00	0.00	6.0	6.8	3.0	10.8
berlin52R-5-3-a	9012	9012	9012	9012	0.18	0.09	0.00	0.00	6.9	8.5	3.7	12.3
berlin52R-5-3-b	8036	8036	8036	8036	0.11	0.00	0.00	0.00	6.7	7.3	3.0	12.3
berlin52R-5-3-c	8224	8224	8224	8224	0.00	0.00	0.00	0.00	6.5	7.4	3.1	13.5
kroA100-R-1-0	21,282	21,282	21,282	21,282	0.00	0.39	0.00	0.00	23.1	95.2	21.5	69.0
kroA100-R-3-0	38,814	38,814	38,814	38,814	0.04	0.16	0.00	0.00	19.9	73.4	14.1	49.3
kroA100-R-3-1	29,264	29,264	29,264	29,264	0.00	1.07	0.00	0.00	23.1	92.6	18.0	60.6
kroA100-R-5-0	50,192	50,192	50,192	50,192	0.15	0.44	0.00	0.00	18.5	63.0	12.1	43.9

Table 5 continued

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
kroA100-R-5-1	35,847	35,847	35,847	35,847	0.38	1.34	0.26	0.00	22.6	73.0	16.6	58.5
kroA100-R-5-3	25,370	25,370	25,370	25,370	0.10	0.89	0.00	0.00	25.7	91.0	19.0	68.4
kroB100-R-1-0	22,141	22,141	22,141	22,141	0.00	0.22	0.00	0.00	23.5	83.6	20.4	72.4
kroB100-R-3-0	37,706	37,706	37,706	37,706	0.15	0.17	0.00	0.00	19.7	69.0	15.7	52.6
kroB100-R-3-1	28,454	28,509	28,454	28,454	0.00	0.50	0.00	0.00	23.1	85.8	19.4	60.5
kroB100-R-5-0	50,964	50,781	50,781	50,781	0.64	0.16	0.00	0.00	19.1	57.8	19.3	46.8
kroB100-R-5-1	35,209	35,209	35,209	35,209	0.10	1.08	0.00	0.00	22.3	81.0	20.5	57.4
kroB100-R-5-3	26,157	26,069	26,069	26,069	0.61	0.52	0.00	0.00	25.6	85.4	23.6	70.1
kroC100-R-1-0	20,749	20,749	20,749	20,749	0.00	0.37	0.00	0.00	23.0	94.6	25.7	70.9
kroC100-R-3-0	37,953	37,953	37,953	37,953	0.38	0.34	0.00	0.00	19.8	86.2	16.2	50.7
kroC100-R-3-1	28,130	28,130	28,130	28,130	0.42	0.62	0.00	0.00	23.6	86.8	18.1	65.1
kroC100-R-5-0	50,085	50,085	50,085	50,085	0.01	0.03	0.00	0.00	18.3	69.1	12.0	46.6
kroC100-R-5-1	33,594	33,594	33,594	33,594	0.49	2.30	1.13	1.37	23.1	73.7	17.0	68.5
kroC100-R-5-3	25,458	25,458	25,458	25,458	0.28	0.51	0.00	0.00	27.1	94.8	21.8	74.2
kroD100-R-1-0	21,294	21,294	21,294	21,294	0.00	0.20	0.00	0.00	22.7	101.5	20.4	66.8
kroD100-R-3-0	38,110	38,110	38,110	38,110	0.09	0.47	0.00	0.00	20.0	72.4	15.4	47.0
kroD100-R-3-1	27,734	27,734	27,734	27,734	0.08	0.55	0.01	0.01	23.8	89.5	18.8	58.5
kroD100-R-5-0	49,100	49,100	49,100	49,100	0.00	0.25	0.06	0.13	20.0	67.1	12.1	41.9

Table 5 continued

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
kroD100-R-5-1	34,246	34,246	34,246	34,246	1.09	0.49	0.00	0.02	23.2	71.6	16.2	54.0
kroD100-R-5-3	25,624	25,624	25,624	25,624	1.55	0.43	0.00	0.00	30.9	86.4	21.7	64.4
kroE100-R-1-0	22,068	22,068	22,068	22,068	0.00	0.28	0.00	0.00	26.6	100.1	21.5	65.4
kroE100-R-3-0	37,935	37,935	37,935	37,935	0.00	0.16	0.00	0.00	21.1	67.8	14.1	46.5
kroE100-R-3-1	29,359	29,359	29,359	29,359	0.07	0.44	0.00	0.00	26.7	90.9	21.7	59.0
kroE100-R-5-0	54,197	54,197	54,197	54,197	0.22	0.16	0.02	0.00	19.0	66.4	12.3	41.6
kroE100-R-5-1	38,359	38,359	38,359	38,359	1.45	0.89	0.20	0.13	25.0	76.6	16.3	55.7
kroE100-R-5-3	27,324	27,256	27,256	27,256	0.25	0.22	0.06	0.03	29.2	109.9	21.2	69.0
kroA200-R-1-0	29,368	29,853	29,368	29,368	0.00	1.08	0.03	0.00	178.8	656.1	186.1	607.8
kroA200-R-3-0	51,539	51,741	51,539	51,539	0.42	0.96	0.03	0.00	123.5	473.4	119.5	380.6
kroA200-R-3-1	37,750	38,208	37,750	37,750	0.62	0.69	0.06	0.00	154.9	574.1	146.3	468.5
kroA200-R-5-0	67,027	67,096	67,027	67,027	0.14	0.99	0.03	0.00	111.1	435.6	100.1	322.0
kroA200-R-5-1	47,912	48,020	47,719	47,719	1.93	1.60	0.33	0.15	158.2	537.1	125.9	407.6
kroA200-R-5-3	33,994	34,195	33,994	33,994	1.24	1.27	0.00	0.00	176.1	495.1	156.1	544.7
kroB200-R-1-0	29,438	29,849	29,438	29,437	0.01	1.01	0.03	0.01	179.3	697.4	191.2	658.7
kroB200-R-3-0	53,597	53,739	53,597	53,597	0.10	0.73	0.12	0.07	122.8	525.5	119.3	443.3
kroB200-R-3-1	38,522	38,943	38,488	38,488	0.27	0.64	0.03	0.01	158.6	606.3	154.6	526.1
kroB200-R-5-0	72,390	72,786	72,357	72,357	0.15	0.31	0.16	0.00	118.3	469.8	102.6	321.8
kroB200-R-5-1	49,944	50,260	49,936	49,927	1.88	1.12	0.74	0.33	150.3	525.9	134.4	405.8
kroB200-R-5-3	36,664	36,926	36,586	36,586	0.84	0.60	0.13	0.07	160.4	652.3	166.8	545.5
Average	29,774.4	29,829.3	29,762.6	29,762.4	0.29	0.46	0.06	0.04	44.4	158.9	39.5	130.1

Table 6 Metaheuristic comparison on the instances of type C

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
berlin52C-1-0-a	7542	7542	7542	7542	0.00	0.00	0.00	0.00	5.1	7.9	3.1	12.6
berlin52C-3-0-a	8144	8144	8144	8144	0.00	0.00	0.00	0.00	5.3	7.6	2.1	8.7
berlin52C-3-0-b	8016	8016	8016	8016	0.00	0.00	0.00	0.00	5.0	6.0	2.1	12.3
berlin52C-3-0-c	9085	9085	9085	9085	0.00	0.07	0.00	0.00	5.0	6.8	2.0	11.3
berlin52C-3-1-a	7952	7952	7952	7952	0.00	0.00	0.00	0.00	6.0	8.6	2.7	10.8
berlin52C-3-1-b	7596	7596	7596	7596	0.00	0.00	0.00	0.00	6.0	7.2	3.2	19.6
berlin52C-3-1-c	7984	7984	7984	7984	0.00	0.00	0.00	0.00	6.0	8.2	3.2	13.2
berlin52C-5-0-a	9430	9430	9430	9430	0.00	0.00	0.00	0.00	5.0	5.9	2.0	10.8
berlin52C-5-0-b	8669	8669	8669	8669	0.00	0.00	0.00	0.00	5.1	5.1	2.0	8.8
berlin52C-5-0-c	9651	9651	9651	9651	0.00	0.00	0.00	0.00	5.1	5.7	2.1	9.1
berlin52C-5-1-a	8811	8811	8811	8811	0.00	0.10	0.00	0.00	6.0	7.2	3.0	10.4
berlin52C-5-1-b	7948	7948	7948	7948	0.00	0.00	0.00	0.00	6.0	6.5	3.0	10.5
berlin52C-5-1-c	8583	8509	8509	8509	0.00	0.00	0.00	0.00	6.0	7.0	3.0	11.6
berlin52C-5-3-a	7907	7907	7907	7907	0.28	0.48	0.00	0.00	6.9	8.8	4.0	16.3
berlin52C-5-3-b	7614	7614	7614	7614	0.00	0.00	0.00	0.00	6.4	7.1	3.0	13.0
berlin52C-5-3-c	7631	7631	7631	7631	0.00	0.00	0.00	0.00	7.0	8.1	4.0	15.0
kroA100-C-1-0	21,282	21,282	21,282	21,282	0.00	0.27	0.00	0.00	23.1	84.9	21.1	77.1
kroA100-C-3-0	24,049	24,049	24,049	24,049	0.00	0.00	0.00	0.00	20.0	67.9	14.0	53.7
kroA100-C-3-1	22,845	23,069	22,845	22,845	0.00	1.51	0.00	0.00	22.0	80.1	18.0	77.6
kroA100-C-5-0	24,745	24,745	24,745	24,745	0.00	0.00	0.00	0.00	18.0	58.5	15.3	50.9
kroA100-C-5-1	22,589	22,589	22,589	22,589	0.00	0.01	0.00	0.00	21.0	84.4	17.6	59.0
kroA100-C-5-3	21,443	21,443	21,443	21,443	0.00	0.00	0.00	0.00	27.7	82.5	25.2	81.6
kroB100-C-1-0	22,141	22,179	22,141	22,141	0.00	0.25	0.00	0.00	23.2	83.1	26.5	82.7

Table 6 continued

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
kroB100-C-3-0	24,887	24,887	24,887	24,887	0.00	0.34	0.00	0.00	20.0	63.5	17.6	59.1
kroB100-C-3-1	22,141	22,141	22,141	22,141	0.30	0.06	0.00	0.00	22.5	77.2	21.0	79.3
kroB100-C-5-0	24,794	24,794	24,794	24,794	0.00	0.00	0.00	0.00	20.3	57.9	15.7	52.8
kroB100-C-5-1	23,159	23,159	23,159	23,159	0.00	0.06	0.00	0.00	23.3	73.4	20.4	68.6
kroB100-C-5-3	22,141	22,141	22,141	22,141	0.00	0.18	0.00	0.00	28.1	79.7	28.5	91.9
kroC100-C-1-0	20,749	20,749	20,749	20,749	0.00	0.18	0.00	0.00	23.0	94.9	24.7	83.1
kroC100-C-3-0	21,340	21,340	21,340	21,340	0.00	0.47	0.00	0.00	20.0	58.0	17.6	58.5
kroC100-C-3-1	20,910	20,910	20,910	20,910	0.00	0.00	0.00	0.00	26.1	92.5	24.1	79.0
kroC100-C-5-0	24,040	24,040	24,040	24,040	0.00	0.00	0.00	0.00	20.6	58.3	16.3	52.9
kroC100-C-5-1	22,827	22,827	22,827	22,827	0.00	0.00	0.00	0.00	22.0	71.8	19.6	64.1
kroC100-C-5-3	21,278	21,278	21,278	21,278	0.07	0.31	0.00	0.00	27.5	91.3	29.1	90.3
kroD100-C-1-0	21,294	21,294	21,294	21,294	0.00	0.02	0.00	0.00	22.8	93.4	25.4	73.3
kroD100-C-3-0	23,809	23,809	23,809	23,809	0.00	0.10	0.00	0.00	19.0	65.3	17.6	51.5
kroD100-C-3-1	21,944	21,944	21,944	21,944	0.00	1.48	0.00	0.00	22.0	87.5	21.0	77.6
kroD100-C-5-0	28,228	28,228	28,228	28,228	0.00	0.02	0.00	0.00	19.0	56.3	16.8	47.4
kroD100-C-5-1	25,102	25,102	25,102	25,102	0.12	0.59	0.00	0.00	21.1	72.7	21.6	57.1
kroD100-C-5-3	21,744	21,744	21,744	21,744	0.01	0.01	0.03	0.00	26.9	94.9	27.2	74.4
kroE100-C-1-0	22,068	22,068	22,068	22,068	0.00	0.36	0.00	0.00	23.0	89.8	25.2	72.5
kroE100-C-3-0	24,383	24,383	24,383	24,383	0.14	0.09	0.00	0.00	19.9	66.5	16.7	53.1
kroE100-C-3-1	22,121	22,121	22,121	22,121	0.00	0.02	0.02	0.01	23.0	76.3	22.3	81.4
kroE100-C-5-0	26,440	26,440	26,440	26,440	0.00	0.01	0.00	0.00	20.0	57.5	16.1	56.5

Table 6 continued

Instance	Best cost				Gap (%)				Runtime (s)			
	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000	ALNS	GILS-RVND	ILS-3000	ILS-10000
kroE100-C-5-1	23,611	23,611	23,611	23,611	0.00	0.20	0.00	0.00	23.6	62.9	19.2	59.8
kroE100-C-5-3	22,455	22,455	22,455	22,455	0.22	0.47	0.00	0.00	27.0	76.5	24.6	72.3
kroA200-C-1-0	29,368	29,737	29,368	29,368	0.00	1.03	0.05	0.00	163.0	641.0	178.8	689.2
kroA200-C-3-0	29,913	29,913	29,913	29,913	0.05	0.59	0.06	0.00	135.2	607.2	111.2	438.9
kroA200-C-3-1	29,368	29,435	29,368	29,368	0.01	1.45	0.04	0.00	148.4	612.4	149.7	526.1
kroA200-C-5-0	32,224	32,224	32,224	32,224	0.01	0.15	0.07	0.02	128.8	404.4	101.7	336.3
kroA200-C-5-1	31,057	31,069	31,057	31,057	0.01	0.49	0.03	0.03	134.6	467.6	131.9	432.6
kroA200-C-5-3	30,023	30,686	30,023	29,994	0.78	0.91	0.14	0.08	152.8	565.1	151.6	481.5
kroB200-C-1-0	29,438	29,790	29,437	29,437	0.02	0.92	0.04	0.01	161.3	628.3	188.9	613.7
kroB200-C-3-0	30,989	30,989	30,989	30,989	0.11	0.93	0.09	0.00	120.5	565.5	112.9	450.0
kroB200-C-3-1	30,442	30,457	30,443	30,442	0.37	1.23	0.02	0.01	131.8	539.3	133.0	524.8
kroB200-C-5-0	37,909	37,909	37,928	37,909	0.01	0.17	0.12	0.09	121.1	401.5	100.4	368.8
kroB200-C-5-1	33,218	33,276	33,309	33,218	1.06	0.49	0.03	0.05	134.0	540.8	119.8	430.8
kroB200-C-5-3	30,028	30,270	30,017	30,017	0.49	0.73	0.10	0.13	161.8	566.0	135.2	497.0
Average	20,639.6	20,673.5	20,640.1	20,637.7	0.07	0.29	0.01	0.01	42.4	153.7	39.4	138.3

with problem-tailored components could be another research direction that is worth following.

Acknowledgements This paper is dedicated to the memory of the last author Vu Hoang Vuong Nguyen, who sadly passed away in 2021.

Funding The authors have not disclosed any funding.

Data availability The data used to support the findings of this study are included in the article.

Declarations

Conflict of interest The authors have not disclosed any competing interests.

References

Ahmed ZH (2014) The ordered clustered travelling salesman problem: a hybrid genetic algorithm. Sci World J 2014:1–13

Alsheddy A (2018) A two-phase local search algorithm for the ordered clustered travelling salesman problem. Int J Metaheir 7(1):80–92

Anily S, Bramel J, Hertz A (1999) 5/3-approximation algorithm for the clustered traveling salesman tour and path problems. Oper Res Lett 24(1):29–35

Applegate DL, Bixby RE, Chvatál V, Cook WJ (2006) The traveling salesman problem: a computational study. Princeton, Princeton University Press

Chisman JA (1975) The clustered traveling salesman problem. Comput Oper Res 2(2):115–119

Dam TT, Thinh Nguyen D, Bui QT, Kien Do T (2019) On the traveling salesman problem with hierarchical objective function. In: 2019 11th international conference on knowledge and systems engineering (KSE), pp 1–5

Doan TT, Bostel N, Hà MH (2021) The vehicle routing problem with relaxed priority rules. EURO J Transp Logist 10:100039

Gavish B, Graves S (1978) The travelling salesman problem and related problems (working paper). In: Operations Research Center, Massachusetts Institute of Technology

Golden BL, Kovacs AA, Wasil EA (2014) Chapter 14: vehicle routing applications in disaster relief. In: Vehicle routing, society for industrial and applied mathematics, pp 409–436

Gutin G, Punnen AP (eds) (2007) The traveling salesman problem and its variations, combinatorial optimization, vol 12. Springer, Boston

Hà MH, Nguyen Phuong H, Tran Ngoc Nhat H, Langevin A, Trépanier M (2022) Solving the clustered traveling salesman problem with d-relaxed priority rule. Int Trans Oper Res 29(2):837–853

Laporte G, Palekar U (2002) Some applications of the clustered travelling salesman problem. J Oper Res Soc 53(9):972–976

Matai R, Singh S, Mittal ML (2010) Traveling salesman problem: an overview of applications, formulations, and solution approaches. In: Davendra D (ed) Traveling salesman problem, theory and applications, IntechOpen, Rijeka, chap 1

Morais VW, Mateus GR, Noronha TF (2014) Iterated local search heuristics for the vehicle routing problem with cross-docking. Expert Syst Appl 41(16):7495–7506

Pacheco T, Martinelli R, Subramanian A, Toffolo TAM, Vidal T (2022) Exponential-size neighborhoods for the pickup-and-delivery traveling salesman problem. Transp Sci 57(2):463–481

Panchamgam K, Xiong Y, Golden B, Dussault B, Wasil E (2013) The hierarchical traveling salesman problem. Optim Lett 7(7):1517–1524

Panchamgam KV (2011) Essays in retail operations and humanitarian logistics. PhD thesis, Robert H.Smith School of Business, University of Maryland (College Park, Md.)

Penna PHV, Subramanian A, Ochi LS (2013) An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. J Heurist 19(2):201–232

Potvin JY, Guertin F (1998) A genetic algorithm for the clustered traveling salesman problem with a prespecified order on the clusters. Springer, Boston, pp 287–299

Reinelt G (1991) TSPLIB: a traveling salesman problem library. ORSA J Comput 3(4):376–384

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472

Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D (2009) Iterated local search for the team orienteering problem with time windows. Comput Oper Res 36(12):3281–3290