The Set Team Orienteering Problem

Tat Dat Nguyen[a], Rafael Martinelli[b], Quang Anh Pham[a], Minh Hoàng Hà[a,*]

[a] *ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam*
[b] *Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro, Brazil*

Abstract

We introduce the Set Team Orienteering Problem (STOP), a generalised variant of the Set Orienteering Problem (SOP), in which customers' locations are split into multiple clusters (or groups). Each cluster is associated with a profit that can be gained only if at least one customer from the cluster is visited. There is a fleet of homogeneous vehicles at a depot, and each vehicle has a limited travel time. The goal of the STOP is to find a set of feasible vehicle routes to collect the maximum profit. We first formulate the problem as a Mixed Integer Linear Programming (MILP) to mathematically describe it. A branch-and-price (B&P) algorithm is then developed to solve the problem to optimality. To deal with large instances, we propose a Large Neighbourhood Search (LNS), which relies on problem-tailored solution representation, removal, and insertion operators. Multiple experiments on newly generated instances confirm the performance of our approaches. The B&P is able to obtain optimal certificates for 77.6% of the STOP instances. Our LNS can achieve optimal solutions for all of these instances except one. More remarkably, we test the algorithms on the SOP via benchmarks available in the literature. It is shown that our B&P can close optimality gaps in 61.2% of these instances. This is the first time such a large number of SOP instances are solved to optimality. Our LNS outperforms existing algorithms proposed to solve the SOP in terms of solution quality. Out of 612 considered instances, it improves 43 best-known solutions.

Keywords: Vehicle routing problem with profits, Set team orienteering problem, Exact and heuristic methods, Branch-and-price, Large neighborhood search

---

*Corresponding author.
Email address: hoang.haminh@phenikaa-uni.edu.vn (Minh Hoàng Hà)

1. Introduction

This paper studies a new extension of the Orienteering Problem (OP), which we call the Set Team Orienteering Problem (STOP), in which customers are split into multiple groups (or clusters). Each cluster is associated with a profit that can be gained only if at least one customer from the cluster is visited. There is a fleet of homogeneous vehicles at the depot, and each vehicle has a limited travel time. The goal of the STOP is to find a set of feasible vehicle routes such that the total collected profit is maximised.

The STOP can be considered a generalised version of the Set Orienteering Problem (SOP), which is first introduced by Archetti et al. (2018). Different from the STOP, the fleet size of vehicles in the SOP is only one. In Archetti et al. (2018), the authors propose a Mixed Integer Linear Programming (MILP) formulation and a matheuristic-based Tabu Search for solving the SOP. Pěnička et al. (2019) introduce an improved MILP formulation for this problem using subtour elimination constraints instead of connectivity cuts like the previous one. They also design a Variable Neighbourhood Search (VNS) and test it on the SOP instances and similar variants, such as the Dubins Orienteering Problem introduced in Pěnička et al. (2017). The authors of Carrabs (2021) employ a Biased Random-Key Genetic Algorithm (BRKGA), which encodes a chromosome as an array of float numbers in a range of [0, 1] and employs three local search procedures for intensification. A graph reduction algorithm based on the work of Gutin and Karapetyan (2009) is also re-used to reduce the problem size, thus improving the computation time of BRKGA. The latest heuristic method proposed to solve the SOP is the Adaptive Memory Matheuristic (AMMH) in Dontas et al. (2023). It employs several local search procedures in combination with the adaptive memory mechanism and a MILP model in a customer insertion-deletion process to intensify the search. The experiments show that the AMMH provides better results than previous methods on the SOP instances introduced in Archetti et al. (2018).

Many routing problems deal with clusters like STOP in the literature. The Clustered Team Orienteering Problem (CTOP) studied in Yahiaoui et al. (2019) shares several common attributes with the STOP. The difference between them is that the profit of a cluster in the CTOP is collected only if all of its customers are served concurrently. The CTOP also has a single-vehicle variant, which is the Clustered Orienteering Problem (COP), introduced in Angelelli et al. (2014). In addition, the Generalised Vehicle Routing Problem (GVRP) in (Ghiani and Improta, 2000; Bektaş et al., 2011; Hà et al., 2014) uses the term *cluster* which represents possible locations of a customer. A set of

vehicles must serve all customers at one of their locations. The goal of the GVRP is to minimise the total travel cost of the vehicles w.r.t. capacity constraints, indicating that the total demand of customers served by a vehicle must not exceed its capacity. A variant of GVRP that has an interesting application in last-mile delivery is the Vehicle Routing Problem with Roaming Delivery Locations (VRPRDL) (Reyes et al., 2017). It models a routing problem by providing *trunk delivery* service, which allows couriers to deliver goods to customers via accessing their trunks (Harbison, 2018). The VRPRDL can be considered the GVRP with time window (TW) constraints, in which TWs of locations in a cluster satisfy special attributes that mimic the itinerary of their corresponding customers. The problem has been solved by various exact methods and metaheuristics: branch-and-price (Ozbaygin et al., 2017), branch-and-price-and-cut (Tilk et al., 2021), large neighbourhood search (Dumez et al., 2021), and hybrid genetic algorithm (Pham et al., 2022).

More recently, Gunawan et al. (2021) introduce a variant of the STOP, known as the Set Team Orienteering Problem with Time Windows (STOPTW), in which a time window is imposed for each customer and the delivery must be done within this time period. The authors propose an Adaptive Large Neighborhood Search algorithm to solve the problem. The preliminary results show the capability of the proposed algorithm to provide good solutions within reasonable computational times compared to the commercial solver CPLEX.

Similar to the SOP, the STOP finds application in the mass distribution of products Archetti et al. (2018). More precisely, we consider the case where multiple customers belong to different supply chains (or clusters) and the profit is associated with each chain rather than with each individual retailer as in the COP. In the STOP, the carrier only has to serve one retailer belonging to each chain with the entire quantity demanded by all of its retailers. Product transportation within a chain will be performed internally. This strategy allows the carrier to reduce the transportation cost, possibly providing a better price for the chains. Another application is when customers in an area want to gather together in a group to place large quantity orders, which may result in a lower price. In this case, a customer in each group will be selected to be the delivery location and receive the entire quantity of all customers in the area. The STOP can also model routing problems in e-commerce where customers can get orders at different addresses, e.g., home, workplace, parents' home, and so on. As such, a cluster in this application includes different delivery locations for a customer. And the profit of a cluster can be set to the profit obtained from serving the corresponding customer.

The main contributions of our work are as follows:

- We propose the STOP, which is a generalised version of several important routing problems with profits. The problem has multiple applications in supply chain distribution and mass distribution products.
- We propose a branch-and-price (B&P) algorithm to solve the problem to optimality. The B&P uses a bucket-based dynamic programming to solve its pricing subproblem and strong branching to reduce the size of the branch-and-bound tree.
- We introduce a metaheuristic that is based on the Large Neighbourhood Search (LNS). The LNS is built from several existing and new components, adapted and designed to efficiently deal with the problem's characteristics.
- We test our proposed methods on the existing SOP instances as well as new STOP instances with up to 1084 customers and 217 clusters. The obtained results clearly show the performance of our approaches. The exact algorithm B&P can close the gap for 77.70% of the STOP instances, while the LNS is able to reach optimal solutions for all but one of these instances.
- For the SOP, 61.2% of the existing instances are solved to optimality (compared with 8.5% in the current literature), and we also provide dual bounds for the first time for those instances. Our metaheuristic has a competitive performance with state-of-the-art methods. More specifically, 43 new best-known solutions are found for the first time in this study. These results are remarkable because our algorithms are designed to mainly solve the multiple-vehicle version.

The remainder of this paper is structured as follows: Section 2 introduces the problem definition and a mixed integer linear programming model. The detailed descriptions of our branch-and-price algorithm and metaheuristic are provided in Sections 3 and 4, respectively. Experimental results are reported in Section 5. And finally, we conclude our work in Section 6.

2. Problem Definition

The STOP is formally defined as follows. Let $G = (V, A)$ be a complete directed graph in which $V = \{0\} \cup C$. Node 0 represents the depot, while C represents the set of customers. Each arc $(i, j) \in A$ is associated with a travel cost c_{ij}. These costs are assumed to satisfy the triangle

inequality and to be symmetric. We also define a set of K clusters $\mathscr{S} = \{S_1, S_2, ..., S_K\}$ where $\bigcup_{k=1}^{K} S_k = V \backslash \{0\}$ and $S_i \cap S_j = \emptyset, \forall i \neq j$. A profit p_k is associated with each cluster S_k and collected only if a customer in cluster S_k is served. Note that, for convenience, we sometimes use S_i to denote the set of customers belonging to the corresponding cluster. The goal of the STOP is to find a set of at most m feasible vehicle routes with maximum total profit serving each cluster at most once. A feasible route must satisfy the following requirements:

- A vehicle departs from the depot and returns to the depot after serving its assigned clusters.
- The total travel cost of each vehicle must not exceed a budget T_{max}.

To mathematically describe the problem, we formulate it as a Mixed Integer Linear Programming (MILP) using the following types of variables:

- y_k: binary variable, equal to 1 if the profit of cluster $S_k \in \mathscr{S}$ is collected by a vehicle, 0 otherwise.
- x_{ij}: binary variable, equal to 1 if arc $(i, j) \in A$ is traversed by a vehicle, 0 otherwise.
- s_i: non-negative real variable, representing the total travel cost of the vehicle when visiting customer i.

The MILP model of the STOP can be written as follows:

$$\max \quad \sum_{S_k \in \mathscr{S}} p_k y_k \tag{1}$$

$$s.t. \quad \sum_{i \in V \backslash \{j\}} x_{ij} = \sum_{i \in V \backslash \{j\}} x_{ji} \qquad \forall j \in V \tag{2}$$

$$\sum_{i \in V \backslash S_k} \sum_{j \in S_k} x_{ij} = y_k \qquad \forall S_k \in \mathscr{S} \tag{3}$$

$$s_j \geq s_i + c_{ij} x_{ij} - T_{max}(1 - x_{ij}) \qquad \forall (i, j) \in A, j \neq 0 \tag{4}$$

$$s_i + c_{i0} x_{i0} \leq T_{max} \qquad \forall i \in V \tag{5}$$

$$\sum_{i \in V \backslash \{0\}} x_{i0} \leq m \tag{6}$$

$$0 \leq s_i \leq T_{max} \qquad \forall i \in V \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{8}$$

$$y_k \in \{0,1\} \qquad \forall S_k \in \mathscr{S} \tag{9}$$

The objective function (1) maximises the total profit collected from all clusters visited by the vehicles. The flow conservation is satisfied by Constraints (2). Constraints (3) represent the relationship between variables y and x, i.e., the profit of a cluster is collected if and only if one customer in that cluster is visited. Constraints (4) compute the total travel cost of vehicles at customers' locations. Constraints (5) ensure that the total travel cost when returning to the depot of each vehicle does not exceed budget T_{max}. The maximum number of vehicles is respected by Constraint (6). Finally, the remaining constraints define the variables' domains.

3. Branch-and-Price

As for many vehicle routing problems, the compact formulation has a small limit on the size of instances it can solve (Martinelli et al., 2014; Baldacci et al., 2011). When pursuing the solution of larger instances, one can resort to decomposition methods that generate exponential-sized formulations, which must be solved using column generation. The formulation presented in this section considers a set of all feasible routes Ω. A route r starts at the depot, visits a non-empty sequence of customers, and ends at the depot. Based on which nodes a route visits, we can define the binary constants a_{kr}, which indicate whether the route visits cluster S_k, and calculate its profit, $p_r = \sum_{S_k \in \mathscr{S}_r} p_k$, where $\mathscr{S}_r$ is the set of clusters visited on route r. Given the binary variables λ_r, which indicate if route r is used in the solution, the formulation is presented as follows:

$$\max \qquad \sum_{r \in \Omega} p_r \lambda_r \tag{10}$$

$$s.t. \qquad \sum_{r \in \Omega} \lambda_r \leq m \tag{11}$$

$$\sum_{r \in \Omega} a_{kr} \lambda_r \leq 1 \qquad \forall S_k \in \mathscr{S} \tag{12}$$

$$\lambda_r \in \{0,1\} \qquad \forall r \in \Omega \tag{13}$$

The objective function (10) maximises the profit obtained by visiting the clusters. Constraint (11) limits the number of vehicles used. Constraints (12) force every cluster to be visited at most once. Constraints (13) present the variables' domains.

The above formulation has an exponential number of λ_r variables. To cope with this issue, the solution of this formulation must be acquired using Column Generation. We will detail the approach in the following sections.

3.1. Pricing

The approach initialises set Ω with a subset of routes (which can be empty) and generates new routes by solving a pricing sub-problem on each iteration. This sub-problem aims at finding a feasible route with the highest reduced cost value. The approach stops when no route with positive reduced costs is found. Given the dual variables γ and β_k associated with Constraints (11) and (12) respectively, the reduced cost of a route r can be defined as in Equation (14).

$$\bar{c}_r = \sum_{S_k \in \mathscr{S}} (p_k - \beta_k)\, a_{kr} - \gamma \tag{14}$$

The pricing problem is then solved by a forward dynamic programming algorithm. Each state of the algorithm is a partial path P starting at the depot and ending at a given customer. All the information associated with a partial path P is represented by a label $\mathscr{L}(P) = \{\bar{c}(P), v(P), t(P), \Pi(P)\}$, where $\bar{c}(P)$ is the path's reduced cost, $v(P)$ the last customer of the path, $t(P)$ the accumulated time, and $\Pi(P)$ the set of visited clusters. The algorithm initially generates the label associated with the depot $\mathscr{L}(P_0) = \{-\gamma, 0, 0, \varnothing\}$, and given a label $\mathscr{L}(P_1) = \{\bar{c}(P_1), v(P_1), t(P_1), \Pi(P_1)\}$ performs the extension to customer i generating label $\mathscr{L}(P_2) = \{\bar{c}(P_1) + p_{\mathscr{K}(i)} - \beta_{\mathscr{K}(i)}, i, t(P_1) + c_{v(P_1)i}, \Pi(P_1) \cup \{i\}\}$, if $t(P_1) + c_{v(P_1)i} + c_{i0} \leq T_{max}$ and $i \notin \Pi(P_1)$, where $\mathscr{K}(i) = k$ if $i \in S_k$. The algorithm stops when no further extension is possible. Next, it tests all extensions back to the depot and returns the routes with positive reduced costs.

In order to reduce the number of labels during the dynamic programming algorithm, we resort to the following dominance rule: Given labels $\mathscr{L}_1(P_1)$ and $\mathscr{L}_2(P_2)$, we say that $\mathscr{L}_1(P_1)$ dominates $\mathscr{L}_2(P_2)$, i.e., $\mathscr{L}_1(P_1) \leq \mathscr{L}_2(P_2)$ if $\bar{c}(P_1) \geq \bar{c}(P_2) \wedge v(P_1) = v(P_2) \wedge t(P_1) \leq t(P_2) \wedge \Pi(P_1) \subseteq \Pi(P_2)$.

We further develop other pricing improvement techniques. The ng-route relaxation (Baldacci et al., 2011) aims to strengthen the dominance rule, allowing non-elementary routes based on memory sets $N_k, \forall S_k \in \mathscr{S}$. Given the minimum travel cost for a pair of clusters S_1 and S_2 as $\min(c_{ij}, \forall i \in S_i, \forall j \in S_2)$, the memory sets are built considering the $\Delta_N = 8$ closest clusters of S_k. Therefore, on the extension from label $\mathscr{L}_1(P_1)$ to label $\mathscr{L}_2(P_2)$, $\Pi(P_2) = \Pi(P_1) \cap N_{\mathscr{K}(v(P_2))} \cup \{v(P_2)\}$.

Another pricing improvement is to consider buckets of labels based on the minimum travel cost, $\varkappa = \min(c_{ij}, \forall (i, j) \in A)$. Thus, we create $\lceil (T_{max} + 1)/\varkappa \rceil$ buckets for each customer and only test the dominance for a pair of labels $\mathscr{L}_1(P_1)$ and $\mathscr{L}_2(P_2)$ when they are in the same bucket, i.e., $\lfloor c(P_1)/\varkappa \rfloor = \lfloor c(P_2)/\varkappa \rfloor$.

3.2. Main Formulation Relaxation

The main formulation (10)–(13) relaxation is solved iteratively, starting with a non-empty Ω set obtained by a simple constructive heuristic and solving the pricing subproblem to obtain new routes. The initial heuristic builds a single solution by choosing on each iteration the cluster with the highest profit, which has not been visited yet, and the vehicle closest to any customer from this cluster, which would not lead to an infeasible route due to T_{max}. When all clusters are visited or no vehicle can perform any further visits, the heuristic stops and returns the solution found.

To aid the Column Generation convergence, we use a simple stabilisation technique as suggested by (Pessoa et al., 2010), with a stabilisation factor $\sigma \in [0, 1]$, used to perform a convex combination of the current dual variable values γ and β_k with the values from the last iteration. The algorithm starts with $\sigma = 0.9$ and, on each mispricing, i.e., when the best route returned by the pricing is not a positive-reduced cost route, the algorithm reduces $\sigma \leftarrow \sigma - 0.1$, until it reaches zero, concluding the stabilisation.

Since the exact pricing presented in Section 3.1 can be time-consuming, on each iteration, the Column Generation first solves a heuristic pricing (Martinelli et al., 2011), which uses a relaxation of the dominance rule, $\mathscr{L}_1(P_1) \preceq \mathscr{L}_2(P_2)$ if $\bar{c}(P_1) \geq \bar{c}(P_2) \wedge v(P_1) = v(P_2)$, which is tested only if the labels are in the same bucket. Thus, the heuristic pricing keeps only one label for each customer's bucket. The Column Generation solves the exact pricing problem only when no positive-reduced cost route is found. If the exact pricing cannot find any positive-reduced cost route, the Column Generation is finished, and the main formulation relaxation is solved.

3.3. Branch-and-Bound

The Column Generation algorithm presented in Section 3.2 only solves the linear relaxation of the main formulation. A Branch-and-Bound approach is used to obtain integer solutions. It solves a Column Generation with branching constraints on each node, resulting in a Branch-and-Price algorithm.

We use two branching rules. The first is a branch on cluster variables y_k, and the branching constraints added in the main formulation are in the form $\sum_{r\in\Omega_k} \lambda_r = 0$ or 1, where Ω_k is the set of routes that visit cluster $S_k \in \mathscr{S}$. The second branching rule is to branch on arc variables x_{ij}, and the branching constraints added are in the form $\sum_{r\in\Omega_{ij}} \lambda_r = 0$ or 1, where Ω_{ij} is the set of routes that traverse arc $(i, j) \in A$.

To further reduce the size of the branching tree, the algorithm conducts strong branching by choosing the ten most fractional branching candidates (clusters or arcs) and evaluating their child nodes. Given the solutions of the two child nodes z_{left} and z_{right}, the strong branching chooses the node with a larger $\alpha \max(z_{left}, z_{right}) + (1 - \alpha) \min(z_{left}, z_{right})$, where $\alpha = 3/4$.

Including branching constraints can make the main formulation infeasible, but this result could be due to a lack of routes in set Ω, an issue the following Column Generation iterations would manage. However, since the formulation result is infeasible, the Simplex algorithm cannot provide dual variable values. To cope with this issue, we include artificial variables in the constraints whenever the formulation is infeasible and perform the first phase of a Two-Phase Simplex algorithm, which uses an artificial objective function. Whenever all artificial variables are null, we remove them from the formulation, restore the original objective function, and usually carry the solution. If the Column Generation finishes with any nonzero artificial variable, the current formulation is infeasible, and the Branch-and-Bound can prune the node.

On each iteration of the Branch-and-Bound, the algorithm should choose a new node to be branched. We use a best-bound strategy, i.e., the node with the highest upper bound value is chosen. This strategy is tentative and aims to reduce the gap on each iteration by pushing down the worst-known upper bound.

3.4. Preprocessing

Before starting the approach, we perform preprocessing to identify incompatible nodes and clusters. Given a customer node $i \in C$, if $c_{0i} + c_{i0} > T_{max}$, node i is incompatible with all other nodes and can then be removed from the problem instance. Given an arc (i, j), if $c_{0i}+c_{ij}+c_{j0} > T_{max}$, then arc (i, j) is incompatible and can also be removed. Moreover, if all customers of a given cluster S_k are incompatible, S_k can be removed from the problem instance. If all arcs between a pair of clusters are incompatible, those two clusters cannot appear on the same route.

4. Metaheuristic

4.1. Large Neighborhood Search framework

The B&P method may not provide good solutions for large STOP instances in a reasonable amount of time. To fill this methodology gap, we develop a metaheuristic based on the Large Neighbourhood Search (LNS) of Shaw (1998). At each iteration, LNS explores a large neighbourhood, which can rearrange a large part of the current solution, thus allowing the search to move to other promising regions of the search space. More precisely, the current solution in LNS is iteratively improved by ruining it (i.e., removing a part of it) and recreating it (i.e., reinserting the removed part in a different way). If the algorithm finds an accepted solution, it becomes the new current solution, and a new large neighbourhood is defined around it. This process is repeated until a stopping criterion (usually a time limit or a maximal number of iterations) is reached.

Algorithm 1: LNS algorithm

```
s ← InitialSolution();
s* ← s;
T ← T_0;
while the stopping criterion is not met do
    Remove ← SelectRemoval();
    Insert ← SelectInsertion();
    s′ ← Insert(Remove(s));
    if Accept(s’, s) then
        s ← s′;
        if f_{s′} > f_{s*} then
            s* ← s′;
        end
    end
    T ← αT;
end
return s*;
```

A STOP solution in our LNS is represented by m vehicle routes. Each route $i \in [1, m]$ is a

permutation of clusters that are visited by vehicle i. This solution representation is also used in (Carrabs, 2021; Pěnička et al., 2019). It allows us to limit the search space around the clusters instead of the customers, which are more numerous. The total profit of visited clusters, i.e., the value of the objective function, of a solution can be trivially computed, but we need to check the solution's feasibility by comparing the cost of every route with the budget T_{max}. As such, a dynamic programming procedure is required to provide the travel cost of each route passing explicit customers. The dynamic programming uses two vectors g and h to track the cost of each route. We denote g_u the minimum travel cost of the partial route from the depot to the customer u. Similarly, h_u is the minimum travel cost of the remaining part of the route from the customer u to the depot. For every two consecutive clusters S_i and S_j in the same route r, we can calculate g and h as follows:

$$g_v = \min_{u \in S_i}(g_u + c_{uv}), \qquad \forall v \in S_j \tag{15}$$

$$h_u = \min_{v \in S_j}(h_v + c_{uv}), \qquad \forall u \in S_i. \tag{16}$$

Then the travel cost of route r can be calculated by:

$$\overline{co}_r = \min_{u \in S_i}(g_u + h_u), \qquad \forall S_i \in r. \tag{17}$$

The pseudo-code of our LNS is illustrated in Algorithm 1. The procedures `SelectRemoval` and `SelectInsertion` randomly select Removal and Insertion operators called `Remove` and `Insert` from the operator pool using the roulette wheel selection. A new solution s' is then generated using these selected operators. The algorithm is stopped after n_{iter} iterations. We evaluate the `Accept` function using the Simulated Annealing (SA) framework, which allows a solution worse than the current one to be accepted. This gives the algorithm the possibility to escape local optima and explore other promising neighbourhoods. We denote f_s as the objective value of solution s. The probability of accepting a new solution s', which is worse than the current solution s, is controlled by the temperature parameter T, as follows:

$$p_{(s',s)} = e^{(f_{s'} - f_s)/T} \tag{18}$$

T is set to exponentially decrease after each iteration by the cooling rate α to lower the probability of the algorithm moving to worse solutions over time. The computation of the initial temperature

T_0 is based on the objective value of the initial solution s_0. It is set so that a new solution that is μ factor worse than the initial solution is accepted with a probability of 50%:

$$T_0 = \frac{-\mu f_{s_0}}{\log_e 0.5} \tag{19}$$

4.2. Removal operators

The removal operators remove a fraction of the clusters from the current solution based on different criteria. Each operator guides the algorithm to a different neighbourhood. The number of clusters to be removed r is based on the size of the current problem. More specifically, it is randomly generated in the range $[2, \max(2, \eta K)]$, where η is the removal control parameter. Each removal operator is associated with a priority weight and is randomly selected using the roulette wheel selection method in each LNS iteration. In this paper, we employ five removal operators as follows:

Random removal. This is the simplest removal proposed in Ropke and Pisinger (2006). It first selects r clusters randomly from the current solution. These clusters are then removed from the solution while keeping the order of the remaining clusters intact. The operator is mainly used to diversify the search, giving every solution a chance to be reached.

Worst-cost removal. The operator is adapted from Ropke and Pisinger (2006). It aims to remove clusters from the solution to reduce the travel cost as much as possible. We denote $\overline{co}$ as the cost of the current solution and $\overline{co}_i$ as the cost of the solution after removing the cluster S_i. The value of $\overline{co}_i$ can be assessed via the vectors g and h defined above. The list of all clusters in the current solution is then sorted in the increasing order of $\overline{co}_i$. The random priority selection process of Ropke and Pisinger (2006) is then applied to select a cluster from the list, such that the clusters at the beginning of the list will be more likely to be selected. A pre-defined parameter, $rp > 1$, which represents the degree of randomness, is used to control the random power of the operator.

Worst-profit removal. The worst-profit removal prioritises removing clusters with a small profit using a similar procedure to the worst-cost removal. The difference is that instead of sorting the cluster list in the order of $\overline{co}_i$, we sort the list in the increasing order of cluster profit p_i.

Worst profit-over-cost removal. The worst profit-over-cost removal balances the impact of profit and cost on the removal process. Similar to the procedure of the worst-cost removal, it prioritises removing clusters based on profit over the cost of the solution after the removal of a cluster. More specifically, we sort the list of clusters in the decreasing order of $\Delta p_i/\overline{co}_i$, where Δp_i is the profit obtained after removing S_i.

Related removal. We adapt this operator from Ropke and Pisinger (2006) with some modifications to the relatedness function. Its main idea is to remove clusters that have high similarities with the expectation of shuffling them around during the insertion phase. The relatedness $R(i, j)$ of two clusters S_i and S_j is given by:

$$R(i, j) = \phi_1 \frac{\sum_{u \in S_i} \sum_{v \in S_j} c_{uv}}{co_{max}.|S_i|.|S_j|} + \phi_2 X(i, j), \tag{20}$$

where the value of $X(i, j)$ is defined as:

$$X(i, j) = \begin{cases} \dfrac{|pos_i - pos_j|}{nc_r}, & \text{if clusters } S_i \text{ and } S_j \text{ are both serviced in route } r \\ 1, & \text{if clusters } S_i \text{ and } S_j \text{ are in different routes.} \end{cases} \tag{21}$$

Here, pos_i is the position of cluster S_i, and nc_r is the number of clusters in route r. The lower value of $R(i, j)$ is, the more related are S_i and S_j. The first term of the Equation (20) measures the average distance for all pairs of customers' locations from the two clusters, S_i and S_j. We normalise the distances by dividing the sum by the maximum possible travel cost between any two customers' locations. The term $X(i, j)$ represents the difference in the relative position of the two clusters in the current solution. If S_i and S_j are serviced by the same vehicle, $X(i, j)$ is correlated with the difference in their positions pos_i and pos_j in the route. Otherwise, $X(i, j)$ is set to 1. These two terms are weighted using the pre-defined parameters ϕ_1 and ϕ_2.

4.3. Insertion operators

Insertion operators are used to guide the algorithm to a good solution in the neighbourhoods created by the removal operators. The partial solutions are reconstructed by selecting unserved clusters and inserting them until no more clusters can be inserted. The new solution after the process must be feasible. We use the vectors g and h to quickly validate the feasibility of the

solution after each insertion. Initially, the solution only includes routes travelling from the starting depot to the ending depot. We initialise the g of the starting depot and the h of the ending depot to 0. When we consider inserting a cluster S_i into the position after the cluster S_j and before the cluster S_k, we can calculate g_u and h_u ($\forall u \in S_i$) using Equations 15 and 16. The new travel cost of the route after the insertion of S_i, $\overline{co}_i$, can then be calculated using the following formula:

$$\overline{co}_i = \min_{u \in S_i}(g_u + h_u). \tag{22}$$

Algorithm 2: Calculate $\overline{co}_i$ when inserting cluster S_i between clusters S_j and S_k.

```
co_i ← ∞;
for u ∈ S_i do
    g_u ← ∞;
    for v ∈ S_j do
        g_u ← min(g_u, g_v + c_vu);
    end
    h_u ← ∞;
    for v ∈ S_k do
        h_u ← min(h_u, h_v + c_uv);
    end
    co_i ← min(co_i, g_u + h_u);
end
return co_i;
```

The procedure of calculating $\overline{co}_i$ for the insertion of S_i is demonstrated in Algorithm 2. The new solution is feasible if and only if the calculated $\overline{co}_i$ is less than or equal to T_{max}. The total complexity of this process is $\mathcal{O}(\kappa^2)$, where κ is the maximum number of customers in a cluster:

$$\kappa = \max_{S_i \in \mathcal{S}} |S_i|. \tag{23}$$

We categorise our insertion operators into two types: predetermined-order insertion and parallel insertion. The former pre-generates a list of unserved clusters beforehand and inserts those clusters in that order. Meanwhile, the latter checks all the clusters during each insertion and determines which cluster to insert. Their detailed descriptions will be discussed in the following sections:

4.3.1. Predetermined-order insertion operators

The insertion procedure of the predetermined order operations is presented in Algorithm 3. In these operators, the order of inserted clusters is predetermined using some cluster-related criteria to define a cluster list, $\mathscr{L}$. Different operators use different criteria. We consider each cluster sequentially and insert it into the best route at the best position possible. More specifically, we find the position to insert such that the increase in the travelling cost, i.e., $\overline{co}_i - \overline{co}$, is the least. The process of calculating $\overline{co}_i$ and finding the best position is demonstrated in Algorithm 2. The complexity of finding the best insertion location for a cluster and inserting it into that position is $\mathscr{O}(|V| * \kappa)$. Thus, the complexity of the whole process in Algorithm 3 is $\mathscr{O}(|V|^2)$. To prevent the search from being trapped in local optima, we employ the same randomised process as in the worst-cost removal with the power parameters rp to select the clusters' order. We illustrate the criteria used to sort the clusters in each operator in the following:

Random order insertion. This operator is adapted from Hammami et al. (2020), in which the order of clusters in $\mathscr{L}$ is fully randomised. Its objective is to increase the diversity of the search.

Profit-based insertion. As proposed by Hammami et al. (2020), this operator sorts clusters' orders based on their profits. Clusters with higher profits are prioritised to be inserted first.

Proximity-based insertion. In this new operator, we sort the order of clusters by a measure named "proximity". For each cluster S_i, its proximity ρ_i is given by:

$$\rho_i = \frac{\sum_{u \in S_i} \sum_{v \in S_i} c_{uv}}{|S_i|^2}. \tag{24}$$

The lower ρ_i is, the closer the customers' locations in the cluster S_i are. The main idea of this operator is that clusters with greater proximity, i.e., those with distant customers, should be inserted first because they are more flexible and could easily change their locations when subsequent clusters are added to reduce the solution cost.

Algorithm 3: Predetermined order insertion.

Input Solution s;
$s' \leftarrow s$;
$\mathscr{L} \leftarrow \{S_i | S_i \in \mathscr{S}, S_i \notin s'\}$;
Sort $\mathscr{L}$ by heuristic criteria;
while $\mathscr{L} \neq \emptyset$ **do**
  $y \leftarrow$ Random[0, 1);
  $id \leftarrow \lfloor y^{rp} |\mathscr{L}| \rfloor$;
  $S_i \leftarrow \mathscr{L}[id]$;
  $\mathscr{L} \leftarrow \mathscr{L} \setminus \{S_i\}$;
  $\Delta\overline{co}_i \leftarrow +\infty$;
  for *each route r in s'* **do**
    for *each insertion position in r* **do**
      Calculate $\overline{co}_i$ for the insertion of S_i into the selected position;
      if $\overline{co}_i \leq T_{max}$ **then**
        $\Delta\overline{co}_i \leftarrow \min(\Delta\overline{co}_i, \overline{co}_i - \overline{co})$;
        Update the best position for S_i;
      end
    end
  end
  if $\Delta\overline{co}_i < +\infty$ **then**
    Insert S_i into the best position;
  end
end
return s';

4.3.2. Parallel insertion operators

Instead of prefixing the order of cluster insertions, parallel insertion operators consider all clusters for insertion in each iteration. The process is demonstrated in Algorithm 4. In each iteration, we evaluate the new costs after inserting each cluster into each possible position, $\overline{co}_i$, and calculate the minimum increased cost of each cluster, S_i, when it is inserted into the best vehicle in the

best position, $\Delta\overline{co}_i$. Thus, the complexity of each insertion operation is $\mathcal{O}(|V|^2)$ and Algorithm 4 runs in $\mathcal{O}(K|V|^2)$. These insertion operators often take much more computational time than the predetermined-order ones. Therefore, they should be used less frequently. As such, the weights of predetermined-order operators will be multiplied by a constant value (8 in our experiments). We only select which cluster to insert, S_{best}, after evaluating the insertion of every cluster into every position of the solution. The way we select S_{best} depends on the different heuristic criteria we choose, as described in the following:

Cheapest insertion. Proposed by Ropke and Pisinger (2006), this is a simple, construction-based greedy heuristic. After the calculation of $\Delta\overline{co}_i$, we select the cluster S_i with the smallest $\Delta\overline{co}_i$ to insert into the position that would increase the total travel cost the least.

Dynamic profit-over-cost insertion. Inspired by Hammami et al. (2020), this insertion operator incorporates the profits of the clusters and the travel cost of the solution. The clusters will be selected in the order of their profit divided by the new cost obtained after inserting them into the solution, i.e., we choose the cluster S_i with the largest value of $p_i/\overline{co}_i$. The clusters are also inserted in the position that would increase the cost the least. This insertion helps the algorithm tackle the profit-related objective function more efficiently.

Regret-k insertion. The idea of this insertion is inspired by the regret insertion operators as suggested in Ropke and Pisinger (2006). We denote $\Delta\overline{co}_i^k$ as the cost increases when inserting the cluster S_i into the k-th best route. The regret-k cost of the cluster S_i is defined as:

$$R_i^k = \sum_{j=1}^{k}(\Delta\overline{co}_i^j - \Delta\overline{co}_i^1). \tag{25}$$

The regret value estimates roughly how much the insertion cost $\Delta\overline{co}_i$ of S_i will increase, or how "difficult" it will be, if we do not insert S_i into the solution immediately. The cluster with the largest regret-k cost is selected to be inserted into the solution in its best position. If any clusters can only be inserted into less than k routes, we will select the cluster that has the fewest feasible insertion routes. The regret-k insertion operators aim to insert the clusters that are harder to insert into the solution first.

Algorithm 4: Parallel insertion.

```
Input Solution s;
s' ← s;
L ← {S_i | S_i ∈ S, S_i not in s'};
while L ≠ ∅ do
    for S_i ∈ L do
        Δco ← +∞;
        for each r in s' do
            for each insertion position in r do
                Calculate co_i for the insertion of S_i into the selected position of vehicle r;
                if co_i ≤ T_max then
                    Δco_i ← min(Δco_i, co_i − co);
                    Update the best position for S_i;
                end
            end
        end
    end
    Select the best cluster S_best based on heuristic criteria;
    L ← L\{S_best};
    if Δco_best < +∞ then
        Insert S_best into the best position;
    end
end
return s';
```

Dynamic profit regret-k insertion. As the regret-k insertions do not consider the profits of clusters, we introduce the dynamic profit regret-k insertion operators. Instead of inserting the cluster with the largest regret-k value, we consider the cluster with the largest product of its profit and regret-k cost, i.e., the cluster S_i with the largest $p_i R_i^k$ to be inserted first. In our experiments, the k values in the two regret insertions are set to $\{2, 3\}$.

4.3.3. Insertion noise factor

Since all our insertion operators always insert a cluster into the position that will increase the cost the least, this can make the algorithm revisit the same solution multiple times. To improve the algorithm diversity, we add some noise to the increased cost calculated in the insertion operators. For each iteration, there is a 50% chance that a noise is added to all the insertion cost $\Delta\overline{co}_i$ calculations. The noise values are generated uniformly randomly in the range $[-\theta.co_{max}, \theta.co_{max}]$, where θ is the noise control parameter.

5. Experimental results

5.1. Benchmark instances and experimental configurations

We first carry out the experiments on the SOP instance benchmark, which is introduced in Archetti et al. (2018).The authors generate two sets of instances called *Set 1* and *Set 2* by modifying the Generalised Travelling Salesman Problem (GTSP) instances of Fischetti et al. (1997). The number of clusters is kept unchanged, and the profit of each cluster is equal to the sum of its customers' profits, which are generated by two strategies, g_1 and g_2. In the first strategy, $g1$, the profit of every customer is set to one, while this figure in the second strategy, g_2 is an integer ranging from 1 to 100. The value of T_{max} in each instance is equal to $\omega.sol_{GTSP}$ where sol_{GTSP} is the objective value of the GTSP best-known solution and ω is set to one of three values $\{0.4, 0.6, 0.8\}$. The number of nodes in this instance class ranges from 52 to 1084, while the number of clusters varies from 11 to 217.

Table 1: General parameters of LNS

Parameter	Symbol	Value	Tuning range
Number of iterations	n_{iter}	20000	-
Removal control	η	0.3	(0.1, 0.5)
Temperature control	μ	0.327	(0.01, 0.4)
Cooling rate	α	0.99948	(0.999, 0.99999)
Noise factor	θ	0.011	(0, 0.2)
Weights	-	-	(0, 4)
Random powers	rp	-	(2, 20)

The benchmark instances for the STOP are not available; we must generate them ourselves. Similar to Yahiaoui et al. (2019), we generate new STOP instances from the SOP instances by dividing the original time limit T_{SOP} by the number of vehicles, i.e., $T_{max} = \left\lceil \frac{T_{SOP}}{m} \right\rceil$. From each SOP instance in Archetti et al. (2018), we derive two new instances with the number of vehicles m equal to two and three.

Table 2: Parameters of the LNS operators

Operator	Weight	Random power (rp)	Others
Random removal	3	-	-
Worst cost removal	3	3	-
Worst profit removal	3	11	-
Worst profit-over-cost removal	1	6	-
Related removal	1	10	$\phi_1 = 18, \phi_2 = 1$
Random order insertion	8×1	-	-
Profit-based insertion	8×4	5	-
Proximity-based insertion	8×1	4	-
Cheapest insertion	1	-	-
Dynamic profit-over-cost insertion	2	-	-
Regret-2 insertion	1	-	-
Regret-3 insertion	3	-	-
Dynamic profit regret-2	4	-	-
Dynamic profit regret-3	3	-	-

All algorithms are implemented in the C++ programming language. We conduct the B&P experiments on an Intel Core i7-8700K @ 3.7GHz with 64GB of RAM and the LNS experiments on an Intel Core i7-7700HQ @ 2.8GHz with 32GB of RAM. The configuration package IRACE (López-Ibáñez et al., 2016) is used to calibrate the parameters of our LNS. The calibration results after running 10,000 experiments of LNS on randomly sampled STOP instances are illustrated in Table 1 (for the general parameters) and Table 2 (for the weights and random degrees of the operators). As mentioned before, the weights of the predetermined-order insertion operators are multiplied by 8 due to their faster running time. All instances and detailed results are reported in

the supplementary material available at http://orlab.com.vn/home/download.

5.2. *Results on the SOP instances*

In this section, we report the experimental results to access the performance of the LNS and the B&P on the existing SOP instances proposed by Archetti et al. (2018). For simplicity, its parameters are set by default, as in the multi-vehicle version, STOP.

5.2.1. *Performance of the branch-and-price algorithm*

We run the B&P with a time limit of six hours (21600 seconds) for each SOP instance, starting the approach with the lower bound found by the LNS, and compare the results with the only exact method available in the literature (Archetti et al., 2018). Table 3 presents the summary results for all configurations (set, g_i, and ω). For each configuration, the first two groups of columns represent the average results for the root node and the complete B&P algorithm. Columns Gap present the average gaps against the best-known solutions, including the ones found in this work, and columns Time present the average times. The last group of columns presents the instance information. Column #Total is the total number of instances in the configuration, and column #Valid is the total number of instances for which our algorithm finds a valid upper bound, i.e., it finishes the root node within the time limit. Note that if the algorithm cannot find a valid upper bound for a given instance, the instance does not contribute to the average gap but contributes to the average time. Finally, columns #Opt and #Lit present the optimal certificates found by B&P and the literature, respectively.

Given the size of larger instances, which could have up to 1084 customers and 217 clusters, it is expected that our B&P algorithm would only be able to solve some of them. Nevertheless, it could not obtain a valid bound for only 19%, and it obtained an optimality certificate for 61.2%, a noteworthy difference from the 8.5% of the current literature. Unfortunately, we cannot perform a more detailed analysis due to the lack of detailed results from the literature. In any case, we provide complete results in the supplementary material.

Table 3: Summary of B&P results on SOP instances.

Set	g	ω	Root		B&P		Instances			
			Gap	Time	Gap	Time	#Total	#Valid	#Opt	#Lit
1	g_1	0.4	0.97	3679.6	0.18	5207.2	51	45	39	4
		0.6	0.88	6109.8	0.38	6931.2	51	39	36	3
		0.8	0.68	8070.3	0.00	8577.9	51	33	33	3
	g_2	0.4	0.85	4058.0	0.11	5606.7	51	44	40	4
		0.6	0.98	6124.8	0.35	7065.1	51	39	36	3
		0.8	0.77	8020.0	0.00	9176.8	51	34	34	3
2	g_1	0.4	5.26	3891.4	2.57	12726.5	51	44	22	6
		0.6	1.60	4251.4	0.98	14314.1	51	43	18	6
		0.8	0.01	4351.1	0.01	4773.6	51	44	43	0
	g_2	0.4	5.64	3782.6	2.73	13735.2	51	44	20	5
		0.6	1.64	4296.9	0.97	14871.8	51	43	16	5
		0.8	0.01	4406.7	0.01	4829.3	51	43	42	0
Total/Avg			1.61	5086.9	0.69	8984.6	612	495	379	42

For each SOP instance, we run the LNS 10 times independently and compare its results with those of the three state-of-the-art methods: VNS (Pěnička et al., 2019), BRKGA (Carrabs, 2021), and AMMH (Dontas et al., 2023). The three algorithms VNS, BRKGA, and LNS are implemented in C++, while AMMH is programmed in C#. The single-thread performance of the machine running the LNS is similar to that of the VNS and the BRKGA, while being 23% poorer than that of the AMMH, according to Passmark Software[1]. Table 4 summarises these comparison results. For each method, we report the following information:

- $\overline{G}$: The average of the *gap* values in percentage over all instances of each instance group. Here, *gap* is calculated as $100 \times \frac{Lit-Best}{Lit}$, where *Best* and *Lit* are the objective values of the best solution found over 10 runs and of the best-known solution in the literature, respectively.
- $\overline{A}$: The average $\overline{gap}$ values in percentage over all instances of each instance group. Here, $\overline{gap}$

[1] https://www.cpubenchmark.net/compare/2906vs897vs3099

is calculated as $100 \times \frac{Lit - Avg}{Lit}$, where Avg is the average objective value over 10 runs.

- #B: The number of best solutions found by a method.

For the LNS, we also report in Column "#O", the numbers of optimal solutions verified by B&P, and in Column "#I", the numbers of new best-known solutions found by our method. The values in bold represent the best values across all methods.

Table 4: Summarized comparison results of different metaheuristics on the SOP instances.

Group			VNS			BRKGA			AMMH		LNS				
Set	g	ω	$\overline{G}$	$\overline{A}$	#B	$\overline{G}$	$\overline{A}$	#B	$\overline{G}$	#B	$\overline{G}$	$\overline{A}$	#B	#I	#O
1	g_1	0.4	0.70	1.27	41	0.23	0.68	40	0.03	46	**-0.06**	**0.01**	**51**	4	**39**
		0.6	1.03	2.15	35	0.44	1.07	36	**0.00**	**47**	0.02	**0.08**	**47**	4	**36**
		0.8	0.69	1.27	34	0.64	1.22	35	0.01	43	**-0.01**	**0.07**	**49**	6	**33**
	g_2	0.4	0.81	1.56	39	0.30	0.62	39	0.02	**49**	**0.00**	**0.07**	48	0	39
		0.6	0.89	2.18	34	0.31	0.85	39	**0.01**	44	**0.01**	**0.08**	**47**	6	**36**
		0.8	0.63	1.32	36	0.61	1.06	33	0.01	43	**-0.01**	**0.07**	**47**	6	**34**
2	g_1	0.4	0.69	1.58	33	1.04	1.61	34	**0.00**	**50**	0.05	**0.26**	43	1	**22**
		0.6	0.48	0.99	35	0.75	1.41	31	0.00	45	**-0.04**	**0.17**	**51**	6	**18**
		0.8	**0.00**	0.02	**51**	0.07	0.22	43	**0.00**	**51**	**0.00**	**0.00**	**51**	0	**43**
	g_2	0.4	0.79	1.64	35	0.78	1.45	32	**0.00**	**48**	0.01	**0.21**	42	2	19
		0.6	0.43	0.98	32	0.50	1.19	31	0.01	42	**-0.02**	**0.16**	**49**	8	**16**
		0.8	**0.00**	0.02	**51**	0.06	0.17	44	**0.00**	**51**	**0.00**	**0.00**	**51**	0	**42**
Total/Avg			0.60	1.25	456	0.48	0.96	437	0.01	559	**0.00**	**0.10**	**576**	43	377

From Table 4, we can see that our LNS outperforms the VNS and BRKGA regarding solution quality. The "Gap" values of LNS are less or equal to those of VNS and BRKGA in all groups. When compared to the AMMH, the LNS outperforms this method in terms of its capability to find the best solutions. The "Gap" values of LNS are better in 6 groups, equal in 3 groups, and worse in only 3 groups. Our LNS finds 576 best-known solutions over 612 instances, outperforming all reference algorithms. These figures for the VNS, BRKGA, and AMMH are: 456, 437, and

559, respectively. In these 576 solutions, there are 43 new best-known solutions that are provided by LNS. Furthermore, the LNS finds 377 out of the 379 optimal solutions found by the B&P. Two instances where LNS cannot reach optimality are 40d198_s2_40_g2 and 80rd400_s1_40_g2. Regarding stability, we only compare the LNS to the VNS and BRKGA since the AMMH does not provide average results over multiple runs. The "Avg" value of the LNS ranges from 0.00% to 0.26%, while these figures for the VNS and the BRKGA can increase up to 2.18% and 1.45%, respectively. This clearly shows that, compared to other methods for the SOP, the LNS can generate high-quality solutions in a more consistent manner.

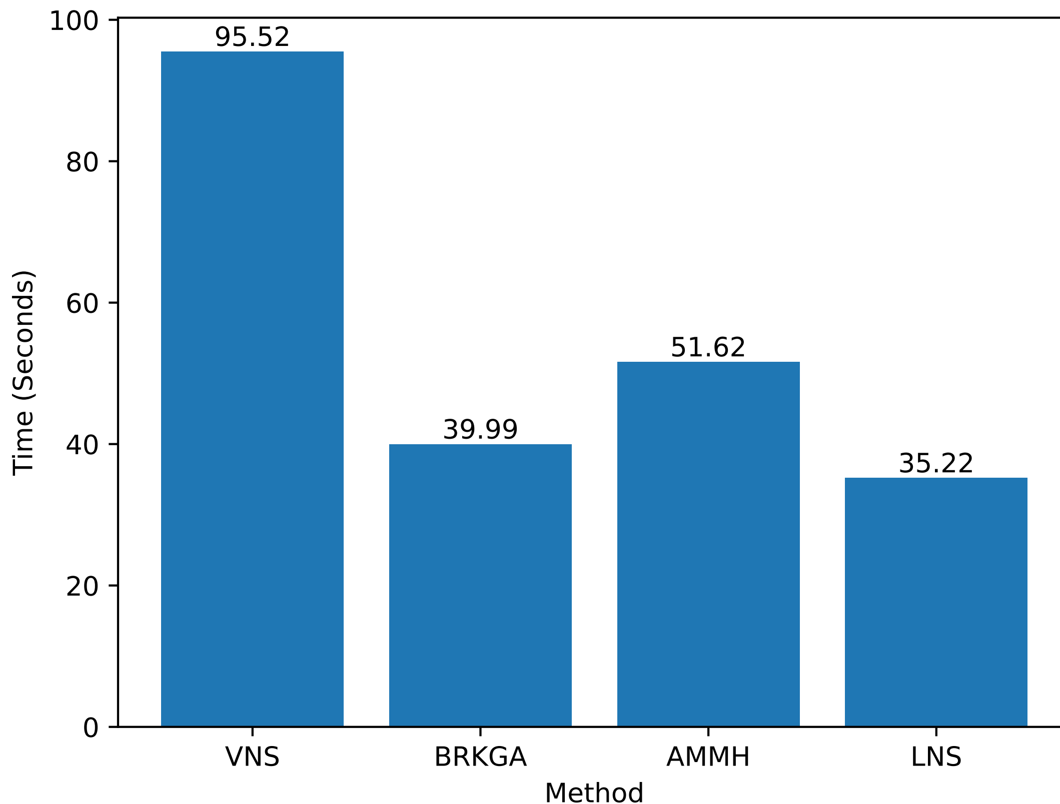

Figure 1: Average running time per SOP instance of each method.

Using the same programming language and being run on machines with similar performance, it could be fair to compare the results of the three algorithms, VNS, BRKGA, and LNS. However,

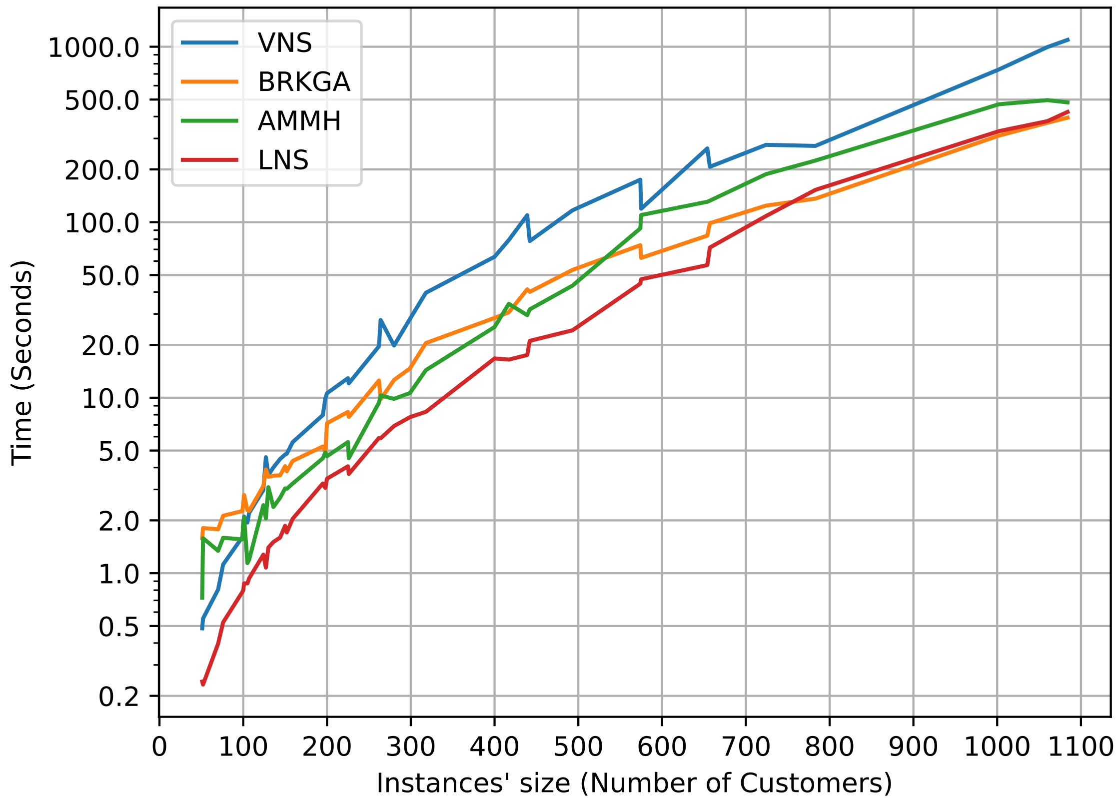

Figure 2: Growth of running time (logarithmic scale) comparison on SOP instances

since it is hard to estimate how much C# is slower than C++, the comparison with AMMH could not be fairly done. Despite that, we still use the raw running time of AMMH in the comparison in an attempt to give a sense of the speed of this method. Figures 1 and 2 compare the running times of the methods. From Figure 1, regarding the running time for each instance, LNS consumes an average of 35.22 seconds, making it the best algorithm in terms of speed. We compare the growth of running time (on a logarithmic scale) by instances' sizes in Figure 2. The LNS is the fastest method on instances with less than 750 customers. For larger instances, the BRKGA is slightly faster. It is worth noting that the BRKGA uses a graph reduction to reduce the computation time by 6.6% to 17.55% on average (Carrabs, 2021). The LNS does not employ this pre-processing procedure. Therefore, we could conclude that LNS is competitive with the existing methods in

terms of computation speed when dealing with large instances.

5.3. Results on the STOP instances

5.3.1. Performance of the branch-and-price algorithm

We present the results for the STOP in Table 5, following almost the same style as Table 3, but including the first column with the number of vehicles and removing the last Column #Lit. From this table, it is possible to note that as the number of vehicles increases, the easier the instance gets, i.e., the gaps and times are smaller and the number of solved instances is larger. It is the expected behaviour since when more vehicles are available, there is a trend towards smaller routes, helping the dynamic programming algorithm used to solve the pricing subproblem.

Regarding the instance parameters, it is possible to note that instances from Set 2 are more challenging than those from Set 1, like the ones with larger ω. However, surprisingly, the root node is faster to solve for some instances with a larger ω, even considering the time limit for those without valid bounds.

The largest instance the B&P is able to solve to optimality has 783 nodes and 157 clusters, and it is so for both Sets 1 and 2, with 2 and 3 vehicles, and for all values of ω. Moreover, the largest instance for which the approach is able to obtain a valid upper bound has 1060 nodes and 212 clusters, for both Sets 1 and 2, but only with 3 vehicles, and for $\omega = 0.4$. On the other hand, the smallest instance in which the B&P is not able to obtain any valid upper bound has 417 nodes and 84 clusters, for Set 1, with 2 vehicles, and for $\omega = 0.8$. Finally, the B&P is the only approach to obtain the best upper bound (also being the optimal solution) for an instance with 99 nodes, 493 clusters, and 2 vehicles. Complete results can be found in the supplementary material.

Table 5: Summary of B&P results the STOP instances.

m	Set	g	ω	Root Gap	Root Time	B&P Gap	B&P Time	Instances #Total	Instances #Valid	Instances #Opt
2	1	g_1	0.4	0.21	1427.0	0.00	1518.0	51	48	48
			0.6	0.45	2819.1	0.09	4494.6	51	46	42
			0.8	1.00	4686.5	0.11	6248.0	51	41	37
		g_2	0.4	0.18	1376.6	0.00	1949.6	51	48	47
			0.6	0.48	2873.1	0.10	5155.6	51	46	39
			0.8	0.95	4635.3	0.12	6882.9	51	41	35
	2	g_1	0.4	2.62	1414.9	0.60	5779.2	51	48	38
			0.6	2.13	3007.2	0.62	9614.6	51	45	31
			0.8	0.96	2037.6	0.54	9608.4	51	48	29
		g_2	0.4	2.49	1409.2	0.61	6047.7	51	48	38
			0.6	2.02	2948.3	0.53	10617.2	51	45	27
			0.8	0.93	2058.8	0.52	9625.5	51	48	29
	Total/Avg			1.20	2557.8	0.32	6461.8	612	552	440
3	1	g_1	0.4	0.13	526.2	0.02	1273.8	51	50	48
			0.6	0.24	1434.8	0.02	2351.3	51	48	46
			0.8	0.34	2124.9	0.04	3957.4	51	48	44
		g_2	0.4	0.13	544.5	0.02	1274.2	51	50	48
			0.6	0.18	1427.8	0.01	2234.9	51	48	47
			0.8	0.31	1898.8	0.07	4263.3	51	48	42
	2	g_1	0.4	0.90	691.8	0.31	3547.7	51	50	43
			0.6	1.57	1426.2	0.41	5733.5	51	48	38
			0.8	0.83	1932.9	0.30	6009.9	51	48	37
		g_2	0.4	0.83	781.4	0.31	3539.9	51	50	43
			0.6	1.37	1416.9	0.36	5620.1	51	48	38
			0.8	0.69	1898.3	0.29	6292.2	51	48	37
	Total/Avg			0.63	1342.0	0.18	3841.5	612	584	511

5.3.2. Performance of the LNS

Similar to the experiments on one-vehicle instances, we run the LNS 10 times on each instance and record the best and average objective values and the average running time. We compare the solutions of the LNS with the best upper bounds provided by the B&P. We report these results in Table 6. The first three columns present the characteristics of the instance groups. For each group, we report the following information:

- Gap: The gap (in percentage) averaged on all instances in a group between the best and average objective values of the solutions found in 10 runs by the LNS.
- Gap_{UB}: The average gap in percentage between the objective value of the best solution found by LNS and the best upper bound provided by B&P. In this calculation, we ignore the instances in which the B&P cannot find a valid upper bound.
- Time: The average running time of the LNS in second.
- #Opt: The number of optimal solutions found by the LNS, as verified by the B&P.

As can be seen in the result table, the LNS can consistently produce high-quality solutions in a reasonable computational time. In 951 instances where the B&P can find optimal solutions, the LNS fails to reach optimality in only one instance. The resultant variations between multiple runs of our metaheuristic are quite small. In more details, the maximal objective gaps of 10 solutions for each instance never bypass 2.67% (for the 2-vehicle instances) and 1.49% (for the 3-vehicle instances). As for computation time, the increment in the number of vehicles only affects the running time of the LNS slightly. The average computation time increases from 35.22 seconds on the one-vehicle instances to 38.03 and 38.38 seconds for the two-vehicle and three-vehicle instances, respectively.

Table 6: Performance of LNS on the STOP instances.

Set	g	ω	$m = 2$				$m = 3$			
			Gap	Gap_{UB}	Time	#Opt	Gap	Gap_{UB}	Time	#Opt
1	g_1	0.4	0.04	0.00	30.94	48	0.02	0.02	24.03	48
		0.6	0.09	0.09	41.30	42	0.02	0.02	39.56	46
		0.8	0.11	0.11	41.83	36	0.05	0.04	39.67	44
	g_2	0.4	0.03	0.00	30.07	47	0.01	0.02	24.18	48
		0.6	0.08	0.10	40.24	39	0.02	0.01	39.02	47
		0.8	0.18	0.12	41.60	35	0.05	0.07	39.13	42
2	g_1	0.4	0.16	0.60	48.04	38	0.09	0.31	50.02	43
		0.6	0.10	0.62	38.91	31	0.04	0.41	44.05	38
		0.8	0.04	0.54	28.43	29	0.05	0.30	33.46	37
	g_2	0.4	0.13	0.61	47.70	38	0.10	0.31	49.58	43
		0.6	0.10	0.53	38.62	27	0.06	0.36	44.23	38
		0.8	0.05	0.52	28.66	29	0.05	0.29	33.66	37
Total/Avg			0.09	0.32	38.03	439	0.05	0.18	38.38	511

6. Conclusions and Future Works

In this paper, we study the Set Team Orienteering Problem (STOP), a multi-vehicle variant of the Set Orienteering Problem (SOP), with multiple real-world applications. To tackle the problem, we propose a branch-and-price (B&P) algorithm and a LNS-based metaheuristic. Our LNS is designed with adapted and new removal and insertion operators to efficiently deal with the profit objective function and the clustering aspect of the problem. The results achieved on benchmark instances of both the problems SOP and STOP show the effectiveness of our methods. We anticipate that our LNS can be utilised to address multiple problems involving profit-related objective functions and/or clustered customers. Thus, applying the LNS to problems such as the CTOP or GVRP could be an interesting research direction. The exact approach is devised by combining a

bucket-based pricing algorithm and a strong branching procedure to obtain optimal solutions and valid upper bounds for large instances. The approach is able to obtain optimal certificates and valid bounds for 77.6% and 92.8% of the STOP instances, respectively, solving instances up to 783 nodes and 157 clusters. The approach can also be used for other prize-collecting and clustering-based routing problems.

Acknowledgments

Rafael Martinelli was partially supported by CAPES (the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Finance Code 001), CNPq (projects 315361/2020-4 and 422470/2021-0), and by FAPERJ (projects E-26/201.417/2022, E-26/010.002232/2019, and E-26/210.041/2023). The manuscript of the paper was finished during the research stay of the corresponding author Minh Hoàng Hà at the Vietnamese Institute for Advanced Studies in Mathematics (VIASM). He wishes to thank this institution for their kind hospitality and support.

References

Angelelli, E., Archetti, C., Vindigni, M., 2014. The clustered orienteering problem. European Journal of Operational Research 238, 404–414.

Archetti, C., Carrabs, F., Cerulli, R., 2018. The set orienteering problem. European Journal of Operational Research 267, 264–272.

Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59, 1269–1283.

Bektaş, T., Erdoğan, G., Røpke, S., 2011. Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. Transportation Science 45, 299–316.

Carrabs, F., 2021. A biased random-key genetic algorithm for the set orienteering problem. European Journal of Operational Research 292, 830–854.

Dontas, M., Sideris, G., Manousakis, E.G., Zachariadis, E.E., 2023. An adaptive memory matheuristic for the set orienteering problem. European Journal of Operational Research 309, 1010–1023.

Dumez, D., Tilk, C., Irnich, S., Lehuédé, F., Péton, O., 2021. Hybridizing large neighborhood search and exact methods for generalized vehicle routing problems with time windows. EURO Journal on Transportation and Logistics 10, 100040.

Fischetti, M., Salazar González, J.J., Toth, P., 1997. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research 45, 378–394.

Ghiani, G., Improta, G., 2000. An efficient transformation of the generalized vehicle routing problem. European Journal of Operational Research 122, 11–17.

Gunawan, A., Yu, V.F., Sutanto, A.N., Jodiawan, P., 2021. Set team orienteering problem with time windows, in: Simos, D.E., Pardalos, P.M., Kotsireas, I.S. (Eds.), Learning and Intelligent Optimization, Springer International Publishing, Cham. pp. 142–149.

Gutin, G., Karapetyan, D., 2009. Generalized traveling salesman problem reduction algorithms. Algorithmic Operations Research 4, 144–154.

Hà, M.H., Bostel, N., Langevin, A., Rousseau, L.M., 2014. An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. Computers & Operations Research 43, 9–19.

Hammami, F., Rekik, M., Coelho, L.C., 2020. A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. Computers & Operations Research 123, 105034.

Harbison, C., 2018. Amazon car delivery: How to get packages delivered to your trunk with new key service. URL: https://www.newsweek.com/amazon-car-delivery-key-trunk-how-cities-home-kit-where-service-899644.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

Martinelli, R., Pecin, D., Poggi, M., 2014. Efficient elementary and restricted non-elementary route pricing. European Journal of Operational Research 239, 102–111.

Martinelli, R., Pecin, D., Poggi, M., Longo, H., 2011. A branch-cut-and-price algorithm for the capacitated arc routing problem, in: Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10, Springer. pp. 315–326.

Ozbaygin, G., Karasan, O.E., Savelsbergh, M., Yaman, H., 2017. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. Transportation Research Part B: Methodological 100, 115–137.

Pěnička, R., Faigl, J., Saska, M., 2019. Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants. European Journal of Operational Research 276, 816–825.

Pěnička, R., Faigl, J., Váňa, P., Saska, M., 2017. Dubins orienteering problem. IEEE Robotics and Automation Letters 2, 1210–1217.

Pessoa, A., Uchoa, E., De Aragão, M.P., Rodrigues, R., 2010. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation 2, 259–290.

Pham, Q.A., Hà, M.H., Vu, D.M., Nguyen, H.H., 2022. A hybrid genetic algorithm for the vehicle routing problem with roaming delivery locations. Proceedings of the International Conference on Automated Planning and Scheduling , 297–306.

Reyes, D., Savelsbergh, M., Toriello, A., 2017. Vehicle routing with roaming delivery locations. Transportation Research Part C: Emerging Technologies 80, 71–91.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40, 455–472.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: International Conference on Principles and Practice of Constraint Programming, Springer. pp. 417–431.

Tilk, C., Olkis, K., Irnich, S., 2021. The last-mile vehicle routing problem with delivery options. OR Spectrum , 1–28.

Yahiaoui, A.E., Moukrim, A., Serairi, M., 2019. The clustered team orienteering problem. Computers & Operations Research 111, 386–399.