# Solving the Park-and-loop Routing Problem by Branch-price-and-cut 

Nicolás Cabrera ${ }^{\text {a }}$, Jean-François Cordeau ${ }^{\text {a }}$, Jorge E. Mendoza ${ }^{\text {a }}$<br>${ }^{a}$ HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T $2 A 7$ Canada


#### Abstract

The park-and-loop routing problem is a variation of the vehicle routing problem in which routes include a main tour that is completed using a vehicle and subtours that are carried out on foot after parking the vehicle. Additionally, the route duration and total walking distance are bounded. To solve the problem, we propose an exact solution method based on the branch-price-and-cut framework. In particular, our method uses problem-specific components to solve the pricing problem. We report on computational experiments carried out on a standard set of 40 instances with up to 50 customers. The results show that our method delivers solutions that compare favorably to existing metaheuristic algorithms, matching all previously best-known solutions and improving 11 of them in reasonable computational times. Moreover, our method provides optimality certificates for 39 out of the 40 instances.


Keywords: vehicle routing problem, branch-price-and-cut, column generation, transportation, park-and-loop

## 1. Introduction

Let $\mathcal{G}=(\mathcal{V}, \mathcal{A})$ be a directed multigraph where $\mathcal{V}$ is the set of vertices and $\mathcal{A}$ denotes the set of directed arcs. The set of vertices comprises a depot 0 and the set of customers $\mathcal{C}=\{1, \ldots, C\}$. Each customer $i \in \mathcal{C}$ has a service time $s_{i}$. There are $k$ homogeneous workers available to serve the customers. These workers can either drive or walk between locations. Accordingly, the set of $\operatorname{arcs} \mathcal{A}=\left\{(i, j)^{m} \mid m=\{d, w\}\right\}$ contains two arcs between each pair of vertices, namely, a driving $\operatorname{arc}(d)$ and a walking $\operatorname{arc}(w)$. Each $\operatorname{arc}(i, j)^{m} \in \mathcal{A}$ has two main attributes: the distance $\delta_{i j}$ and the time $\eta_{i j}$. Each worker has a maximum daily walking distance $\zeta$ and is hired for a full working day
lasting $\phi$ units of time. The maximum distance that a worker may walk between two points is $\theta$. The fixed cost of hiring a worker is $c^{f}$. Additionally, there is a variable cost associated with driving ( $c^{v}$ per unit of distance). The park-and-loop routing problem (PLRP) consists in finding a set of least cost routes (starting and ending at the depot) while ensuring that: each customer is served precisely once; the total duration of each route does not exceed the working day duration; and the distance walked by any worker does not exceed the limit. Because it is a generalization of the well-known vehicle routing problem (VRP), the PLRP is NP-hard.

In order to serve the set of customers three types of routes can be designed: pure vehicle routes performed by a worker driving between customers; pure walking routes performed by a worker walking between customers; and finally park-and-loop routes that are vehicle routes with walking subtours. Figure 1 shows a feasible solution to a small PLRP instance with 6 customers. In this solution, customers 1 and 2 are served by a worker driving a vehicle (i.e., a pure vehicle route), while customers 3 to 6 are served by a worker following a park-and-loop route. More specifically, the worker leaves (i.e., parks) the vehicle at the depot and walks to serve customer 3 . The worker then walks back to the depot to pick up the vehicle and drives to customer 4. After serving customer 4 , the worker parks the vehicle and walks to serve customer 5. The worker then walks back to customer 4. Finally, the worker drives to customer 6 and then returns to the depot.


Figure 1: PLRP instance and solution example.

The PLRP is closely related to the two-echelon last-mile delivery problem (2E-LMDP) discussed by Martinez-Sykora et al. (2020); a variant of the traveling salesman problem (TSP) consisting in finding a single park-andloop route to serve a set of customers. The authors propose an exact branch-and-cut algorithm capable of solving instances with up to 30 customers. As opposed to the PLRP, the number of walking subtours starting at a given parking spot is limited to one. More recently, Reed et al. (2022) introduced the capacitated delivery problem with parking (CDPP). This problem extends the 2E-LMDP by allowing the worker to perform an unlimited number of walking subtours starting at the same parking spot. The authors propose a mixed integer programming (MIP) formulation that can solve instances with up to 50 customers. In addition, they describe a two-step heuristic that can handle instances with up to 100 customers. However, the authors limit the number of customers in a walking subtour to three. In the PLRP, the number of customers in a walking subtour is not constrained.

The PLRP is also related to the truck and trailer routing problem (TTRP) introduced by Semet (1995). This variant of the VRP considers a fleet of trucks pulling trailers to serve a set of customers. The problem also considers a set of decoupling locations (i.e., parking places), where trailers can be detached as some of the customers are only accessible by the truck without the trailer. Most of the work on the TTRP has focused on heuristic algorithms (Chao, 2002; Sheuerer, 2006; Lin et al., 2009; Villegas et al., 2011; Derigs et al., 2013; Villegas et al., 2013). These methods are capable of providing high quality solutions for instances with up to 150 customers. In the TTRP, routes are constrained by the combined capacity of the truck and the trailer. In contrast, routes in the PLRP are constrained by the maximum walking distance and the working day duration.

In the last decade, researchers have turned their attention to developing exact methods for more restricted TTRP variants (Belenguer et al., 2016; Parragh \& Cordeau, 2017; Rothenbächer et al., 2018). Belenguer et al. (2016) propose a branch-and-cut algorithm for the single truck and trailer routing problem with satellite depots (STTRPSD). This algorithm is capable of optimally solving instances with up to 50 customers when limiting the number of parking places to 10 . Parragh \& Cordeau (2017) propose a branch-and-price algorithm to solve the truck and trailer routing problem with time windows (TTRPTW). Their method is capable of solving instances with up to 100 customers. Rothenbächer et al. (2018) propose a branch-price-and-cut algorithm that outperforms the algorithm of Parragh \& Cordeau (2017) on the
same TTRPTW variant. Their method is capable of finding the optimal solution for 35 additional instances. As opposed to the TTRPTW, customers in the PLRP do not have an associated time window. Therefore, the latter can be expected to be more difficult to solve using branch-price-and-cut.

Another related problem is the doubly open park-and-loop routing problem (DOPLRP) introduced by Cabrera et al. (2022). This variant of the VRP consists in finding a set of least-cost routes that may start and end at any customer. To solve the DOPLRP, the authors propose a two-phase matheuristic called MSH. This approach was capable of handling instances with up to 3,800 customers. As opposed to the PLRP, the time duration of each walking subtour is bounded. More recently, Le Colleter et al. (2023) defined the park-and-loop routing problem with parking selection (PLRP-PS). The main difference with other problem variants is that the vehicle can only be parked at dedicated parking locations. To solve the PLRP-PS, the authors introduce a small and large neighborhood search methaeuristic (SLNS). The authors use new specific techniques to select parking spots that significantly speed up the algorithm. Their algorithm was capable of providing solutions for instances with up to 400 customers and 352 dedicated parking spots.

The closest problem to the PLRP is the VRP with transportable resources (VRPTR) introduced by Coindreau et al. (2019). In the VRPTR, a set of workers has to serve a set of customers. The workers can either walk or drive to their next location and are allowed to carpool (i.e., share a vehicle). To solve their VRPTR, the authors use a mixed integer linear program (MILP). Their experiments show that their MILP can only solve instances with up to 18 customers. Thus, they also propose a variable neighborhood search (VNS) algorithm. This method can solve instances with up to 50 customers with a running time limit of 10 hours. The authors also studied a version of their problem in which carpooling is not allowed. The latter perfectly matches the PLRP definition. Le Colleter et al. (2023) reported results benchmarking VNS, MSH, and SLNS on the Coindreau et al. (2019) instances. Their study sets SLNS as the state-of-the-art algorithm since it unveiled eight new bestknown solutions. Note, however, that neither of these methods is exact. Moreover, none of those approaches has been assessed with respect to a lower bound. In other words, no optimality gaps have been reported for the solutions they provide.

The contribution of this article is two-fold. From a methodological perspective, we propose a branch-price-and-cut algorithm to solve the PLRP. The key algorithmic component of our method is the pulse algorithm used
to solve the pricing problem. The latter extends the procedure introduced in Lozano et al. (2015) to handle the park-and-loop structure of the routes and the inclusion of the subset row inequalities proposed by Jepsen et al. (2008). In addition, we present a set of acceleration strategies tailored to the PLRP. From a computational perspective, we perform extensive experiments on the set of 40 instances introduced by Coindreau et al. (2019), arguably the most widely used testbed for VRPs with park-and-loop structure. Our algorithm is the first to prove optimality for 39 of the instances. In addition, we developed an online web application गthat allows researchers to visualize and download the best-known solutions for the PLRP. They can also upload their solutions for plotting and checking.

This paper is organized as follows. Section 2 presents the mathematical formulation. Section 3 describes the proposed branch-price-and-cut algorithm. Section 4 presents the acceleration strategies that crucially improve the algorithm's performance. Section 5 contains the computational experiments. Finally, Section 6 presents the conclusions and outlines potential paths for future research.

## 2. Problem formulation

We define a route $r$ as an ordered set of directed arcs starting and ending at the depot. The customers served in the route are represented by the set $\mathcal{C}_{r}$. Let the subsets $\mathcal{A}_{r}^{d}$ and $\mathcal{A}_{r}^{w}$ contain the driving and the walking arcs in $r$ respectively. A route $r$ is time-feasible if

$$
\begin{equation*}
\sum_{(i, j) \in \mathcal{A}_{r}^{d} \cup \mathcal{A}_{r}^{w}} \eta_{i j}+\sum_{i \in \mathcal{C}_{r}} s_{i} \leq \phi . \tag{1}
\end{equation*}
$$

Similarly, a route $r$ is walking-feasible if

$$
\begin{equation*}
\sum_{(i, j) \in \mathcal{A}_{r}^{w}} \delta_{i j} \leq \zeta . \tag{2}
\end{equation*}
$$

The cost $c_{r}$ of a route is equal to the sum of the variable and the fixed cost, that is,

$$
\begin{equation*}
c_{r}=\sum_{(i, j) \in \mathcal{A}_{r}^{d}} \delta_{i j} c^{v}+c^{f} \tag{3}
\end{equation*}
$$

[^0]Let $\mathcal{R}$ be the set of all feasible routes and let $a_{i r}$ be a parameter that takes the value 1 if and only if route $r \in \mathcal{R}$ serves customer $i \in \mathcal{C}$. Finally, let $x_{r}$ be a binary variable equal to 1 if route $r \in \mathcal{R}$ is selected and 0 otherwise. A set covering (SC) formulation for the PLRP can be stated as follows:

$$
\begin{equation*}
\min \sum_{r \in \mathcal{R}} x_{r} c_{r} \tag{4}
\end{equation*}
$$

subject to

$$
\begin{array}{rlrl}
\sum_{r \in \mathcal{R}} a_{i r} x_{r} & =1 & \forall i \in \mathcal{C} \\
\sum_{r \in \mathcal{R}} x_{r} & \geq\left\lceil\frac{\sum_{i \in \mathcal{C}} s_{i}}{\phi}\right\rceil & & \\
\sum_{r \in \mathcal{R}} x_{r} & \leq k & & \\
x_{r} & \in\{0,1\} & \forall r \in \mathcal{R} . \tag{8}
\end{array}
$$

The objective function (4) minimizes the total cost. Constraints (5) ensure that all customers are served exactly once. Constraints (6) and (7) provide, respectively, a lower and an upper bound on the number of routes used to serve the set of customers. Constraints (8) are the domain restrictions. Note that the number of feasible routes $|\mathcal{R}|$ grows exponentially. Thus, solving SC by enumerating all the feasible routes is usually not possible. As an alternative, the SC can be solved using a branch-price-and-cut algorithm which is described next.

## 3. Solution method

In this section, we present an exact branch-price-and-cut algorithm (BPC) to solve the SC model. A BPC algorithm is a branch-and-bound algorithm in which, at each node of the enumeration tree, the linear relaxation of an integer formulation is solved using column generation (CG) and tightened by adding valid inequalities (i.e., cuts). For completeness, Section 3.1 describes the column generation algorithm. Section 3.2 defines the pricing problem. Section 3.3 describes the key strategies used in the pricing problem algorithm. Section 3.4 presents the valid inequalities. Finally, Section 3.5 describes the branching rules.

### 3.1. Column generation

CG is a solution method that can solve integer programs with a large number of decision variables (i.e., columns). To achieve this goal, two optimization problems are solved iteratively: A master problem that considers only a small subset of variables and a pricing problem that generates promising variables to be added to the master problem. If, at a given iteration, no variables are added to the master problem, the CG algorithm ends. We refer the interested reader to the book by Desaulniers et al. (2006) and the study by Lübbecke \& Desrosiers (2005) for a review of techniques and applications of column generation.

In our case, the MP corresponds to the relaxed version of formulation (4)-(8). The relaxed set covering problem (RSCP) considers only a subset of feasible routes (columns) $\overline{\mathcal{R}} \subseteq \mathcal{R}$ and is obtained by relaxing the integrality constraints on the $x_{r}$ variables. Let $\pi_{i} \in \mathbb{R}, \sigma \geq 0$, and $\rho \leq 0$ be the dual variables associated with constraints (5), (6), and (7) respectively. The reduced cost of variable $x_{r}$ is then given by the following expression:

$$
\begin{equation*}
\mathfrak{r}_{r}=c_{r}-\sum_{i \in \mathcal{C}} a_{i r} \pi_{i}-\sigma-\rho . \tag{9}
\end{equation*}
$$

Having $\min _{r \in \overline{\mathcal{R}}}\left\{\mathfrak{r}_{r}\right\} \geq 0$ ensures that the RSCP is solved optimally. If there exists a route $r$ with $\mathfrak{r}_{r}<0$, we add the corresponding variable $x_{r}$ to the subset $\overline{\mathcal{R}}$. Therefore, the goal after solving the RSCP is to identify a route with negative reduced cost. This problem is referred to as the pricing problem and is the topic of the next section.

### 3.2. Pricing problem

Let $\mathcal{G}^{\prime}=\left(\mathcal{V}^{\prime}, \mathcal{A}^{\prime}\right)$ be a directed multigraph, henceforth referred to as the modified network, where $\mathcal{V}^{\prime}$ is the set of nodes including a start depot $\underline{0}$ and an end depot $\overline{0}$. Thus, $\mathcal{V}^{\prime}=\mathcal{C} \cup\{\underline{0}, \overline{0}\}$ and $\mathcal{A}^{\prime}=\mathcal{A}^{1} \cup \mathcal{A}^{2} \cup \mathcal{A}^{3} \cup \mathcal{A}^{4} \cup \mathcal{A}^{5}$, where

- $\mathcal{A}^{1}=\left\{(i, j)^{d} \mid i \in \mathcal{C} \cup\{\underline{0}\}, j \in \mathcal{C}\right\}$ is the set of driving arcs arriving to any customer,
- $\mathcal{A}^{2}=\left\{(i, j)^{w} \mid i \in \mathcal{C} \cup\{\underline{0}\}, j \in \mathcal{C}\right\}$ is the set of walking arcs arriving to any customer,
- $\mathcal{A}^{3}=\left\{(i, \overline{0})^{d} \mid i \in \mathcal{C}\right\}$ is the set of driving arcs going from any customer to the end depot,
- $\mathcal{A}^{4}=\left\{(i, \underline{0})^{w} \mid i \in \mathcal{C}\right\}$ is the set of walking arcs going from any customer to the start depot, and
- $\mathcal{A}^{5}=\left\{(\underline{0}, \overline{0})^{w}\right\}$ is a fictitious arc connecting the start and end depots.

The attributes of the arc connecting $\underline{0}$ and $\overline{0}$ are set to zero. Note that $\operatorname{arcs} \mathcal{A}^{4}$ are necessary in order to allow workers to start walking subtours from the depot. Figure 2 shows how the routes in Figure 1b are mapped to the modified network.


Figure 2: Route representation on the modified network.
As mentioned previously, the objective of the pricing problem is to find routes from $\underline{0}$ to $\overline{0}$ with a negative reduced cost. With this in mind, we must define the reduced cost of each arc in $\mathcal{G}^{\prime}$. The reduced cost of an arc $(i, j)^{m} \in \mathcal{A}^{\prime}$ is defined as:

$$
r_{i j}= \begin{cases}\delta_{i j} c^{v}-\pi_{j}, & (i, j)^{m} \in \mathcal{A}^{1}  \tag{10}\\ -\pi_{j}, & (i, j)^{m} \in \mathcal{A}^{2} \\ \delta_{i j} c^{v}, & (i, j)^{m} \in \mathcal{A}^{3} \\ 0, & (i, j)^{m} \in \mathcal{A}^{4} \\ 0, & (i, j)^{m} \in \mathcal{A}^{5}\end{cases}
$$

In terms of network flows, the start depot offers one unit of flow that is demanded by the end depot. Thus, the formulation of the pricing problem
partially follows that of the shortest path problem in which the weight of the arcs corresponds to their reduced cost. Nevertheless, in the context of the PLRP, a node in graph $\mathcal{G}^{\prime}$ can only be visited once (unless the node is used as a parking spot), which means that paths are pseudo-elementary. Moreover, paths (i.e., routes) are constrained by two resources: the time $\phi$ and the walking distance $\zeta$. Hence, the pricing problem corresponds to solving an (elementary) shortest path problem with resource constraints and park-and-loop (SPPRC-PL). The pricing problem itself is NP-hard. However, several algorithms exist for solving the elementary shortest path problem with resource constraints. The interested reader is referred to the article by Pugliese \& Guerriero (2013) for a review of exact algorithms for solving resource-constrained shortest path problems. In this paper, we adapt the pulse algorithm (PA) proposed by Lozano et al. (2015) for solving the elementary shortest path problem with resource constraints (ESPPRC). This algorithm has been successfully used as a component to solve other hard combinatorial optimization problems (Restrepo et al., 2012; Montoya et al., 2016; Lozano \& Smith, 2017; Arslan et al., 2018; Schrotenboer et al., 2019).

The PA is a recursive algorithm based on the idea of propagating pulses through the network from a start node (i.e., the start depot) $\underline{0}$ to an end node (i.e., the end depot) $\overline{0}$. While traversing the network node by node, the pulse builds a partial path $\mathcal{P}$ that includes all the nodes already visited. Additionally, the pulse contains information on the attributes associated with the path, such as the cumulative reduced cost or the resource consumption. Whenever a pulse reaches the end node $\overline{0}$ it contains all the information of a feasible path $\mathcal{P}$ from $\underline{0}$ to $\overline{0}$. If the path has a negative reduced cost, it can be added to the subset $\overline{\mathcal{R}}$ and the best solution can be updated.

The PA ensures that the optimal path $\mathcal{P}^{*}$ is always found by implicitly enumerating all paths from $\underline{0}$ to $\overline{0}$. However, one can easily truncate the PA to accelerate the search by solving the problem heuristically, as it is discussed in Section 4.2. To prevent the PA from explicitly enumerating all possible paths, the algorithm uses a set of pruning strategies. These strategies allow the PA to stop (prune) the propagation of a partial path as soon as there is enough evidence that the partial path will not improve the current best solution or that it will lead to an infeasible solution. Note that stopping a partial path from propagating allows for discarding a large number of paths, as it discards all the paths that begin with it. Thus, the earlier a partial path is stopped, the better. This idea is shared with other algorithms, like branch-and-bound, where an implicit enumeration is performed. Similarly,
the strength of the pulse algorithm depends on the pruning strategies. In what follows the terms stopping or pruning a path are used interchangeably.

Algorithm 1 presents the main logic of the pulse algorithm. Line 1 initializes the partial path. Lines 2 to 4 initialize the reduced cost and the cumulative resource consumption. Line 5 runs the bounding procedure given the bound step size $\Delta$ and the bounding time limits $[\bar{t}, \underline{t}]$. Line 6 extends a pulse at the start node. Finally, line 7 returns the optimal path. Note that in the case where reaching the end node $\overline{0}$ is not possible due to the resource constraints (i.e., the pricing problem is infeasible), the optimal path ends up empty.

```
Algorithm 1 pulseSearch function
Require: \(\mathcal{G}^{\prime}\), directed multi graph; \(\phi\), duration limit; \(\zeta\), walking distance
    limit; \(\underline{0}\), start node; \(\overline{0}\), end node; \(\Delta\), bound step size; \([\bar{t}, \underline{t}]\), bounding time
    limits.
Ensure: \(\mathcal{P}^{*}\), optimal path.
    \(\mathcal{P}^{*} \leftarrow \emptyset\)
    \(r(\mathcal{P}) \leftarrow 0\)
    \(t(\mathcal{P}) \leftarrow 0\)
    \(w(\mathcal{P}) \leftarrow 0\)
    bound \(\left(\mathcal{G}^{\prime}, \Delta,[\bar{t}, t]\right) \quad \triangleright\) see 33.3 .2
    pulse \((\underline{0}, r(\mathcal{P}), t(\mathcal{P}), w(\mathcal{P}), \mathcal{P}) \quad \triangleright\) see Algorithm 2
    return \(\mathcal{P}^{*}\)
```

Algorithm 2 shows the recursive procedure pulse, where $\Gamma_{w}^{+}(i)$ corresponds to the set of arcs leaving node $i$ on foot and $\Gamma_{d}^{+}(i)$ by driving. Lines 1 to 4 use the pruning strategies, namely, infeasibility, bounds, rollback, and path completion, to try to prune a partial path. If the pulse is not pruned, line 5 adds the node to the partial path. From lines 6 to 10 , the algorithm recursively propagates the pulse by driving to node $j$. From lines 11 to 17 , the algorithm recursively propagates the pulse using Algorithm 3 by walking to node $j$, thus parking the vehicle at node $i$.

```
Algorithm 2 pulse function
Require: \(i\), current node; \(r\), cumulative reduced cost; \(t\), cumulative time;
    \(w\), cumulative walking distance; \(\mathcal{P}\), partial path.
    if \(\neg\) feasibility \((i, t(\mathcal{P}), w(\mathcal{P}))\) then \(\quad \triangleright\) see 3.3 .1
        if \(\neg\) bounds \((i, r(\mathcal{P}), t(\mathcal{P}))\) then \(\quad \triangleright\) see \$3.3.2
            if \(\neg\) rollback \((i, r(\mathcal{P}), t(\mathcal{P}), w(\mathcal{P}), \mathcal{P})\) then \(\quad \triangleright\) see §3.3.3
                if \(\neg\) complete_path \((i, r(\mathcal{P}), t(\mathcal{P}), w(\mathcal{P}), \mathcal{P})\) then \(\triangleright\) see \$3.3.4
                    \(\mathcal{P}^{\prime} \leftarrow \mathcal{P} \cup\{i\}\)
                    for \(j \in \Gamma_{d}^{+}(i)\) do
                            \(r\left(\mathcal{P}^{\prime}\right) \leftarrow r(\mathcal{P})+r_{i j}\)
                                    \(t\left(\mathcal{P}^{\prime}\right) \leftarrow t(\mathcal{P})+\eta_{i j}+s_{j}\)
                                    pulse \(\left(j, r\left(\mathcal{P}^{\prime}\right), t\left(\mathcal{P}^{\prime}\right), w(\mathcal{P}), \mathcal{P}^{\prime}\right)\)
                    end for
                        for \(j \in \Gamma_{w}^{+}(i)\) do
                    \(r\left(\mathcal{P}^{\prime}\right) \leftarrow r(\mathcal{P})+r_{i j}\)
                                    \(t\left(\mathcal{P}^{\prime}\right) \leftarrow t(\mathcal{P})+\eta_{i j}+s_{j}\)
                                    \(w\left(\mathcal{P}^{\prime}\right) \leftarrow w(\mathcal{P})+\delta_{i j}\)
                                    \(p s \leftarrow i\)
                                    pulse_parked \(\left(j, r\left(\mathcal{P}^{\prime}\right), t\left(\mathcal{P}^{\prime}\right), w\left(\mathcal{P}^{\prime}\right), p s, \mathcal{P}^{\prime}\right)\)
                    end for
                end if
            end if
        end if
    end if
    return void
```

Algorithm 3 shows the recursive procedure pulse_parked. Lines 1 to 3 try to prune the partial path using the infeasibility, bounds, and rollback pruning strategies. If the pulse is not pruned, line 4 adds the node to the partial path. From lines 5 to 22 , the algorithm recursively propagates the pulse by walking to node $j$. At line 9 , we check if the path is returning to the parking spot. If so, at line 14 the algorithm checks if it is possible to prune the partial path by using the subtour fixing strategy. If the partial path is not pruned, the algorithm recursively propagates the pulse using Algorithm [2.

```
Algorithm 3 pulse_parked function
Require: \(i\), current node; \(r\), cumulative reduced cost; \(t\), cumulative time;
    \(w\), cumulative walking distance; \(p s\), parking spot; \(\mathcal{P}\), partial path.
    if \(\neg\) feasibility \((i, t(\mathcal{P}), w(\mathcal{P}))\) then \(\quad \triangleright\) see \(\$ 3.3 .1\)
        if \(\neg\) bounds \((i, r(\mathcal{P}), t(\mathcal{P}))\) then \(\quad \triangleright\) see \$3.3.2
            if \(\neg \operatorname{rollback}(i, r(\mathcal{P}), t(\mathcal{P}), w(\mathcal{P}), \mathcal{P})\) then \(\quad \triangleright\) see \(\$ 3.3 .3\)
                \(\mathcal{P}^{\prime} \leftarrow \mathcal{P} \cup\{i\}\)
                for \(j \in \Gamma_{w}^{+}(i)\) do
                        \(r\left(\mathcal{P}^{\prime}\right) \leftarrow r(\mathcal{P})+r_{i j}\)
                        \(t\left(\mathcal{P}^{\prime}\right) \leftarrow t(\mathcal{P})+\eta_{i j}+s_{j}\)
                        \(w\left(\mathcal{P}^{\prime}\right) \leftarrow w(\mathcal{P})+\delta_{i j}\)
                if \(p s=j\) then
                    \(t\left(\mathcal{P}^{\prime}\right) \leftarrow t\left(\mathcal{P}^{\prime}\right)-s_{j}\)
                    if \(j \in \mathcal{C}\) then
                                    \(r\left(\mathcal{P}^{\prime}\right) \leftarrow r(\mathcal{P})+\pi_{j}\)
                    end if
                    if \(\neg\) subtour_fixing \((j, t(\mathcal{P}), \mathcal{P})\) then \(\quad \triangleright\) see 3.3 .5
                        pulse \(\left(j, r\left(\mathcal{P}^{\prime}\right), t\left(\mathcal{P}^{\prime}\right), w\left(\mathcal{P}^{\prime}\right), \mathcal{P}^{\prime}\right)\)
                    end if
                else if \(p s \neq j\) then
                    pulse_parked \(\left(j, r\left(\mathcal{P}^{\prime}\right), t\left(\mathcal{P}^{\prime}\right), w\left(\mathcal{P}^{\prime}\right), p s, \mathcal{P}^{\prime}\right)\)
                        end if
                end for
        end if
        end if
    end if
    return void
```

The following section provides further detail regarding the pruning strategies used by the PA.

### 3.3. Pruning strategies

In Sections 3.3.1, 3.3.2, and 3.3.3 we describe the adaptation of the original PA pruning strategies proposed by Lozano et al. (2015), namely, infeasibility, bounds, and rollback pruning. In Section 3.3.4 we present the path completion strategy (line 4 in Algorithm 2) adapted from Cabrera et al. (2020) which was used to solve the constrained shortest path problem (CSP). Finally, in Section 3.3.5 we describe a new pruning strategy for
the PA specifically tailored for our pricing problem called subtour fixing (line 16 in Algorithm (3).

### 3.3.1. Infeasibility pruning

The intuition of this pruning strategy is to stop a pulse as soon as it becomes evident that it will not be able to reach the end node $\overline{0}$ while meeting the resource constraints. Thus, we can safely stop a partial path $\mathcal{P}$ from propagating if any of the following conditions holds:

- $t(\mathcal{P})>\phi ;$
- $w(\mathcal{P})>\zeta$.

Discarding infeasible partial paths is often used as a key strategy to improve the performance of labeling algorithms. Note that this strategy could easily be extended to include other route constraints such as the presence of time windows at customer locations or a time limit on each walking subtour.

### 3.3.2. Bounds pruning

Similar to the infeasibility pruning strategy, we can stop a pulse from propagating if there is enough information to prove that the current partial path will not lead to improving the best solution found so far. More specifically, if there is evidence that the partial path will not be able to decrease the best objective function $r\left(\mathcal{P}^{*}\right)$ we can stop the partial path from propagating. With this purpose, we use a bounding scheme that computes lower bounds $\underline{r}(i, t(\mathcal{P}))$ for every node $i \in \mathcal{G}^{\prime}$ and for a set of possible values of time resource consumption $t(\mathcal{P})$. More specifically, these bounds contain the minimum reduced cost from any node $i$ to the end node $\overline{0}$ given a partial resource consumption $t(\mathcal{P})$.

To compute the lower bounds, we solve a SPPRC-PL from every node $i \in \mathcal{G}^{\prime}$ to $\overline{0}$ given a time consumption of $t(\mathcal{P})=\bar{t}-\Delta$. These problems are overly-constrained as the pulse only has $\Delta$ units of time available to reach the end node $\overline{0}$. Accordingly, the pulse algorithm can easily solve these problems to optimality. Each of the solutions found is a valid lower bound on the minimum reduced cost that can be obtained from node $i$ given a time consumption $t(\mathcal{P}) \geq \bar{t}-\Delta$. After finding these bounds, we proceed to solve a SPPRC-PL from every node $i \in \mathcal{G}^{\prime}$ to $\overline{0}$ given a time consumption of $t(\mathcal{P})=$ $\bar{t}-2 \Delta$. Although the resulting problems are less constrained, we already have vital information for partial paths with a time consumption between
$[\bar{t}-\Delta, \bar{t}]$. We continue with this procedure, solving SPPRC-PL problems with $\{\bar{t}-3 \Delta, \bar{t}-4 \Delta, \ldots, \underline{t}\}$. Note that at the end of this procedure we have a lower bound for every node and every discrete time step between $\underline{t}$ and $\bar{t}$. With this information, we can prune a partial path $\mathcal{P}$ if $r(\mathcal{P})+\underline{r}(i, t(\mathcal{P})) \geq r\left(\mathcal{P}^{*}\right)$.

For further details regarding this strategy, the reader is referred to Lozano et al. (2015). Note that in this section we describe the bounding scheme using only the time consumption $t(\mathcal{P})$. In preliminary experiments, we considered computing additional bounds with respect to the walking distance $w(\mathcal{P})$. However, this did not lead to a major improvement in performance.

### 3.3.3. Rollback pruning

The choice between a depth-first search (DFS) and a breadth-first search (BFS) strategy has been widely studied in the literature as it affects the performance of every labeling setting/correcting algorithm. Although it is possible to affect the behavior of the pulse algorithm to make a BFS exploration through the usage of pulse queues as presented in Cabrera et al. (2020), in this article we consider a version of the pulse algorithm that follows a pure DFS strategy.

In some cases this behaviour could lead to exploring vast unpromising regions of the search space, before backtracking to correct poor decisions made earlier. To overcome this problem, the rollback pruning strategy reevaluates the last choice made. More specifically, consider a partial path $\mathcal{P}_{\underline{0}, i}$ from $\underline{0}$ to $i$ that is extended to node $l$ and then reaches node $j$. Once the partial path $\mathcal{P}_{\underline{0}, j}=\mathcal{P}_{\underline{0}, i} \cup l \cup j$ reaches node $j$, we check if visiting node $j$ before node $l$ is a better alternative. If so, we can stop the pulse from propagating. In practice, we check if $r\left(\mathcal{P}_{\underline{0}, j}^{\prime}\right) \leq r\left(\mathcal{P}_{\underline{0}, j}\right)$ and $t\left(\mathcal{P}_{0, j}^{\prime}\right) \leq t\left(\mathcal{P}_{\underline{0}, j}\right)$ to prune path $\mathcal{P}_{0, j}$, where path $\mathcal{P}^{\prime}$ corresponds to the path that skips node $l$.

### 3.3.4. Path completion

One of the main drawbacks of the PA, as it was presented by Lozano et al. (2015), is that the best path $\mathcal{P}^{*}$ is only updated when the end node $\overline{0}$ is reached. Thus, the main purpose of the path completion strategy is to update the best path (primal bound) at intermediate nodes in the network.

To do so, we take advantage of the information computed in the bounding procedure presented in Section 3.3.2. Formally, let us consider a partial path $\mathcal{P}_{0, i}$ arriving to node $i$. The path completion strategy adds the minimum reduced cost path $\mathcal{P}_{i, \overline{0}}^{r t}$ given a time consumption $t\left(\mathcal{P}_{\underline{0}, i}\right)$ to the partial path
$\mathcal{P}_{\underline{0}, i}$, that is, $\mathcal{P}_{\underline{0}, \overline{0}}=\mathcal{P}_{\underline{0}, i} \cup \mathcal{P}_{i, \overline{0}}^{r t}$. If the completed path is feasible and the reduced cost is lower than the current primal bound, we update the incumbent solution accordingly. Furthermore, we can stop the incoming partial path $\mathcal{P}_{0, i}$ from propagating, because (by construction) we know that the complete path is already the minimum reduced cost path stemming from this partial path given the current time consumption. This procedure is used in Line 4 of Algorithm 2 and is adapted from the path completion strategy proposed by Cabrera et al. (2020) for the constrained shortest path problem.

### 3.3.5. Subtour fixing

Consider a partial path $\mathcal{P}_{\underline{0}, j}$ from $\underline{0}$ to $j$ in which $j$ is used as a parking spot. Moreover, consider a walking subtour visiting three or more customers $S=\left\{i_{1}, i_{2}, \ldots, i_{|S|}\right\}$ and stemming from node $j$. The number of possible walking subtours visiting all the customers in $S$ and using $j$ as a parking spot can be calculated as $|S|$ !. Thus, during the recursive search, the algorithm can visit $j$ to retrieve the vehicle following several different sequences in which all the customers in $S$ are visited. The total reduced cost associated to each of these sequences is equal (i.e., $\sum_{i \in S}-\pi_{i}$ ), while the total time may differ. Note that a slower walking subtour may result in the impossibility of visiting other customers using the partial path $\mathcal{P}_{\underline{0}, j}$.

Once again, as the algorithm follows a pure DFS strategy, is possible that many partial paths will be explored before backtracking and correcting the sequence followed for visiting the customers in $S$. For this reason, every time a partial path completes a walking subtour we check if it is possible to stop that path from propagating. In practice, we follow two steps. First, we check if it is the first time (since the BPC algorithm started) that the customers in $S$ are visited by a walking subtour. If so, we solve a traveling salesman problem minimizing the total walking time and store the value $t^{*}(S)$ in memory. If not, we retrieve the value stored previously. Second, we compare the current time of the subtour $t(S)$ with the best time $t^{*}(S)$. If $t(S)>t^{*}(S)$ we can stop the pulse from propagating and thus avoid exploring paths that use an inefficient walking subtour.

### 3.4. Valid inequalities

The optimal solution of the RSCP can be fractional. In that case before applying branching decisions we first try to improve (lift) the lower bound. To do so, we draw upon valid inequalities (cuts). Particularly, we include the subset row inequalities proposed by Jepsen et al. (2008) for subsets of
three customers. These inequalities have been used in different applications as the multi-depot vehicle routing problem (Contardo \& Martinelli, 2014), the vehicle routing problem with time windows (Costa et al., 2019), and the two-echelon capacitated vehicle routing problem (Marques et al., 2020). The subset row inequalities with $|\mathcal{S}|=3$ are defined as

$$
\begin{equation*}
\sum_{r \in \mathcal{R}}\left\lfloor 1 / 2 \sum_{i \in \mathcal{S}} a_{i r}\right\rfloor x_{r} \leq 1, \quad \forall \mathcal{S} \subseteq \mathcal{C} \tag{11}
\end{equation*}
$$

Each of these inequalities has an associated dual variable $\beta_{\mathcal{S}} \leq 0$. These inequalities ensure that for a given subset $\mathcal{S} \subseteq \mathcal{C}$ the number of routes serving two or more customers is less or equal to 1 .

To separate these inequalities we enumerate all customer triplets and check if the inequality is violated in the current optimal solution. It has been observed by multiple researchers that even if subset row inequalities tend to have a positive impact on the quality of the lower bound, they significantly increase the complexity of solving the pricing problem. Thus, in line with Jepsen et al. (2008), we allow our BPC algorithm to add up to $\varphi$ at each iteration with a minimum violation of $\varepsilon$. Inequalities with a greater violation are given priority. In our BPC , we set $\varphi$ to 1 and $\varepsilon$ to 0.1 .

Note that adding these inequalities modifies the definition of the reduced cost of a route. More specifically, if we denote $\mathcal{S}$ as the subset of triplets of customers for which the subset row inequality has been generated and added to the master problem, then the reduced cost of a route is defined as

$$
\begin{equation*}
\mathfrak{r}_{r}=c_{r}-\sum_{i \in \mathcal{C}} a_{i r} \pi_{i}-\sigma-\rho-\sum_{\mathcal{S} \in \mathcal{S}} \beta_{\mathcal{S}}\left\lfloor\frac{1}{2} \sum_{i \in \mathcal{S}} a_{i r}\right\rfloor . \tag{12}
\end{equation*}
$$

Adding a subset row inequality implies that a penalty of $\beta_{\mathcal{S}}$ must be paid if two or more customers of the corresponding triplet are served by the route. To account for this term using the PA, we add a new resource for each subset in $\mathcal{S}$ that stores the number of times that a customer in the subset has been visited. If, while extending a pulse this value reaches a value of 2 we subtract the $\beta_{\mathcal{S}}$ from the cumulative reduced cost. The reader should note that all the pruning strategies outlined in Section 3.3 can still be used without any changes. This is an advantage compared to algorithms that rely
on assessing dominance between two partial paths (labels) in which extending the dominance criteria is required.

In addition, to strengthen the linear relaxation of the SC formulation, we lifted constraint (6) using the value of the objective function of the minimum spanning tree on the graph $\mathcal{G}$, as follows:

$$
\begin{equation*}
\left\lceil\frac{\sum_{i \in \mathcal{C}} s_{i}+M S T(\mathcal{G})}{\phi}\right\rceil \leq \sum_{p \in \mathcal{P}} x_{p} \tag{13}
\end{equation*}
$$

Constraint (13) ensures that the number of routes in the solution of the RSCP is at least the minimum number of routes needed to serve all the customers. Lifting this constraint does not have any impact on the pricing problem structure.

### 3.5. Branching rules

Adding the inequalities outlined in Section 3.4 does not guarantee that the optimal solution of the RSCP will be integral. In the case in which the optimal solution of the RSCP is still fractional, we resort to branching on the arc flow variables. To do so, we define $a_{i j r}^{m}$ as the number of times an arc $(i, j)^{m} \in \mathcal{A}$ appears in route $r$. Then, for each $\operatorname{arc}(i, j)^{m} \in \mathcal{A}$ it is possible to compute the number of times it appears in a solution as

$$
\begin{equation*}
b_{i j}^{m}=\sum_{r \in \overline{\mathcal{R}}} a_{i j r}^{m} . \tag{14}
\end{equation*}
$$

We select for branching the arc for which the value of $b_{i j}^{m}$ is closest to 0.5 . In the case of a tie, we prioritize driving arcs.

Branching on an arc implies creating two child nodes for the branch-andprice tree: one child in which the arc is forbidden (the value is set to zero) and one child in which the arc is fixed (the value is set to one). These conditions are enforced locally in the pricing problem by modifying the graph $\mathcal{G}^{\prime}$. If an arc must be forbidden, the arc is simply removed from graph $\mathcal{G}^{\prime}$. However, fixing an arc is not as straightforward as it depends on the transportation mode. If the arc $(i, j)$ is a driving arc, we remove all the driving arcs starting from node $i$ and ending at any node different than $j$. Moreover, we remove all the driving arcs ending at node $j$ that start at any node different than $i$. In addition, we remove the walking $\operatorname{arcs}(i, j)^{w}$ and $(j, i)^{w}$. If the $\operatorname{arc}(i, j)$ is a walking arc, we remove all the walking arcs starting from node $i$ and ending
at any node different than $j$. We also remove all the walking arcs ending at node $j$ that start at any node different than $i$. In addition, we remove the driving $\operatorname{arcs}(i, j)^{d}$ and $(j, i)^{d}$.

## 4. Acceleration strategies

We now describe several ideas that we use to speed up our BPC algorithm.

### 4.1. Dual stabilization

Usually, CG-based algorithms suffer from slow convergence, a phenomenon called the tailing-off effect (Desaulniers et al., 2006). An important technique for alleviating this issue is to implement a dual stabilization method. In this work, we implemented the $\alpha$-schedule procedure presented by Pessoa et al. (2013). This procedure aims to correct the values of the dual variables used to solve the pricing problem based on previous dual solutions. Algorithm 4 shows the pseudocode of the procedure. Line 1 initializes the value of $l$. Line 2 initializes the value of the dual variables. Line 3 updates the smoothing factor. Line 4 computes the value of the dual variables that will be used for solving the pricing problem. Line 5 updates the value of $l$. Line 6 calls the pricing problem solved with the procedure described in Section 3.2. If mispricing occurs the algorithm returns to line 3. Otherwise, the number of iterations is updated, and the master problem is solved once again.

```
Algorithm \(4 \alpha\)-scheduling function
Require: \(\alpha\), smoothing factor.
    \(l \leftarrow 1\)
    \(\pi^{0} \leftarrow \pi^{i n}\)
    \(\bar{\alpha} \leftarrow[1-l(1-\alpha)]^{+}\)
    \(\pi^{\text {sep }}=\bar{\alpha} \pi^{0}+(1-\bar{\alpha}) \pi^{o u t}\)
    \(l \leftarrow k+1\)
    Call the pricing problem with \(\pi^{\text {sep }}\)
    if Mis-pricing occurs then
        Go to step 3
    else
        \(t \leftarrow t+1\)
        Solve the master problem
        Go to step 1
    end if
```


### 4.2. Heuristic pulse algorithm

The pricing problem does not have to be solved to optimality at every iteration of the CG. Thus, it is a common practice to design heuristics to quickly find promising solutions in the first iterations of the CG (Desaulniers et al., 2006). In our case, we can easily truncate the PA to solve the pricing problem heuristically by imposing a stopping criterion. More specifically, we can heuristically stop the PA from propagating more pulses if the number of paths found with negative reduced cost reaches $\Upsilon$. Moreover, we can impose a time limit $\Lambda$. Then, if the CPU time for solving the pricing problem reaches $\Lambda$ and the PA has already found a promising path, we stop the PA.

Furthermore, note that by allowing the PA to find paths with a park-and-loop structure, the complexity of solving the pricing problem increases heavily. However, it is possible that in some iterations of the CG, paths without subtours may have a negative reduced cost. Accordingly, we adopt a leveled pricing strategy in which we first run the PA without considering the walking arcs. Only if the algorithm was not able to find promising paths do we proceed to run the PA by allowing walking subtours.

### 4.3. Initialization

It is well known that the performance of a CG algorithm is affected by the initial set of columns. Usually, including a high-quality set of initial columns can help the algorithm perform a better estimation of the dual variables associated with the RSCP constraints. Although one could initialize the pool of columns using $|\mathcal{C}|$ routes visiting one customer, in our implementation we use the output of the sampling phase of the MSH matheuristic proposed by Cabrera et al. (2022). For the sake of completeness we briefly describe the procedure here.

Algorithm 1 presents the main logic of the sampling phase of MSH. Line 1 initializes the set of initial routes $\overline{\mathcal{R}}$. Line 2 initializes the iteration number. From lines 4 to 13 , the algorithm populates $\overline{\mathcal{R}}$ using a set of TSP heuristics $\mathcal{H}$ and the splitting procedure split $\langle\cdot, \cdot\rangle$. Line 4 randomly selects a TSP heuristic $h$ from $\mathcal{H}$. Line 5 generates a giant route $\tau^{t}$ visiting all customers using $h$. Line 6 generates a solution, denoted as $s^{t}$. Line 7 joins the routes in solution $s^{t}$ to set $\overline{\mathcal{R}}$. Lines 8-12 update the incumbent solution. Line 15 returns the set of initial routes that will be used by our BPC.

```
Algorithm 5 MSH sampling function
Ensure: initial solution \(\overline{\mathcal{R}}\)
    \(\overline{\mathcal{R}} \leftarrow \emptyset\)
    \(t \leftarrow 1\)
    while \(t<T \wedge\) samplingTime \(<Q\) do
        \(h \leftarrow \mathcal{H}\)
        \(\tau^{t} \leftarrow h(\mathrm{G})\)
        \(s^{t} \leftarrow \operatorname{split}\left\langle\mathcal{G}, \tau^{t}\right\rangle\)
        \(\overline{\mathcal{R}} \leftarrow \overline{\mathcal{R}} \cup s^{t}\)
        if \(\mathrm{t}=1\) then
            \(s^{*} \leftarrow s^{t}\)
        else if \(f\left(s^{t}\right)<f\left(s^{*}\right)\) then
            \(s^{*} \leftarrow s^{t}\)
        end if
        \(t \leftarrow t+1\)
    end while
    return \(\overline{\mathcal{R}}\)
```

Require: G, graph; $\mathcal{H}$, heuristic set; $T$, iteration $\operatorname{limit} ; Q$, time limit.

The key algorithmic component of the sampling phase of MSH is the split $\langle\cdot, \cdot\rangle$ procedure used to extract a solution $s^{t}$ from the giant TSP-like tour in Line 6. The split procedure follows two steps. In the first step, it constructs a directed acyclic graph defined by a set of nodes $\mathrm{N}=\left(v_{0}, v_{1}, \ldots, v_{i}, \ldots, v_{n}\right)$ and the set of arcs A . Node $v_{0}$ is a dummy node, while nodes numbered 1 to $n$ represent the customer in the $i$-th position of the giant tour $\tau^{t}$. Each arc $(i, j) \in \mathrm{A}$ represents a feasible route $r\left(v_{i+1}, v_{j}\right)$ visiting customers $\left(v_{i+1}, \ldots, v_{j}\right)$. To evaluate if an arc should be added to G (line 12) it solves a subproblem that can be seen as a multi-resource version of the single truck and trailer routing problem with satellite depots (STTRPSD) proposed in Villegas et al. (2010). The solution to this subproblem yields a route with one or more walking subtours. In the second step, the split procedure finds the shortest path from $v_{0}$ to $v_{n}$ in G. The set of arcs (i.e., routes) along the shortest path corresponds to a feasible solution $s^{t}$.

In our BPC algorithm we use the default parameters of the MSH. Namely, the number of iterations is set to 2,500 , and the time limit is set to 60 seconds. For further detail regarding the MSH, the reader is referred to Cabrera et al. (2022).

## 5. Computational experiments

In this section, we present the computational experiments that we performed on a set of standard instances from the literature. Our goal is to analyze the performance of the proposed BPC algorithm and its main components. In addition, we describe a web application that can be used to check the quality of the solutions found by any researcher working on the PLRP or related variants. The BPC algorithm was implemented in Java using the jORLib2 2 library and compiled using Java 1.8.0_331. The experiments were performed on an Intel core i7 @2.30 GHz Quad-Core processor with 12GB of RAM. We used CPLEX 20.1 to solve the SC formulation and the RSCP. Due to the randomness induced by the initialization procedure described in Section 4.3, for every experiment, we ran five replicates. On each replicate we used a different value from the set $\{1,2,3,4,5\}$ to seed Java's pseudorandom number generator. This way we ensure consistency across the initial solutions used by each algorithm and validate the consistency of the proposed BPC. We also set a time limit of 2 hours for every run.

### 5.1. Test instances

To assess the efficiency and effectiveness of the BPC algorithm, we use the set of instances proposed by Coindreau et al. (2019) for the VRPTR without carpooling. Each instance considers a number of customers $n$ in the set $\{20,30,40,50\}$ located inside a square grid of 10 km by 10 km . The depot is located at the center of the grid. The distance between nodes (customers and depot) is the Euclidean distance. In addition, to compute driving and walking times they consider a driving speed of $30 \mathrm{~km} / \mathrm{h}$ and a walking speed of $4 \mathrm{~km} / \mathrm{h}$. The customer service times range from 20 to 35 minutes. The maximum daily walking distance for each worker is 5 km and the day duration is 7 hours (i.e., 420 minutes). It is worth recalling that any customer location can be used as a parking spot. For each instance, Coindreau et al. (2019) fixed the number of available vehicles as the number of routes in the corresponding VRP solution. The fixed cost is set to $0 \$$ and the variable cost is set to $1 \$ / \mathrm{km}$. The objective is to minimize the total cost. An instance is referred to as " $\mathrm{n} \mathrm{A}_{-} \mathrm{i}$ ", where n stands for the number of customers, A represents the size of the used time window (i.e., all day), and i denotes the instance unique identifier. A total of 10 instances are considered for each instance size (i.e.,

[^1]number of customers). After fine tuning, we set the bound step size in the PA $(\Delta)$ to 15 . The bounding time limits $[\underline{t}, \bar{t}]$ are set to $[120,420]$. In addition, the maximum number of paths $\Upsilon$ is set to 10 and the time limit $\Lambda$ to 15 seconds. Finally, the parameter $\alpha$ used to stabilize the values of the dual variables is set to 0.8 .

### 5.2. Assessing the BPC performance

In this section, we analyze the performance of the proposed BPC and compare it to that of the state-of-the-art algorithms for the PLRP, namely, the VNS introduced by Coindreau et al. (2019), the MSH designed by Cabrera et al. (2022), and the SLNS developed by Le Colleter et al. (2023).

Table 1 compares the performance of the proposed branch-price-and-cut algorithm in the best performing replicate against the benchmark algorithms in terms of solution quality. Each row corresponds to an instance size. Columns 2, 5, 8, and 11 show the number of best-known solutions (BKSs) found by each algorithm. Columns 3, 6, 9, and 12 report the average gap with the best-known solution. Columns 4, 7, 10, and 13 show the maximum gap with the best-known solution.

Table 1: Solution quality on the Coindreau et al. (2019) instances.

| $\|\mathcal{C}\|$ | VNS |  |  | MSH |  |  | SLNS |  |  | BPC |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | $\overline{\text { BKSs }}$ | Avg. $\Delta$ | Max. $\Delta$ | BKSs | Avg. $\Delta$ | Max. $\Delta$ | BKSs | Avg. $\Delta$ | Max. $\Delta$ | BKSs | Avg. $\Delta$ | Max. $\Delta$ |
| 20 | 1/10 | 4.09\% | 8.87\% | 10/10 | 0.00\% | 0.00\% | 10/10 | 0.00\% | 0.00\% | 10/10 | 0.00\% | 0.00\% |
| 30 | 0/10 | 3.13\% | 5.89\% | 7/10 | 0.01\% | 0.04\% | 8/10 | 0.00\% | 0.04\% | 10/10 | 0.00\% | 0.00\% |
| 40 | 3/10 | 1.23\% | 5.31\% | 6/10 | 0.15\% | 0.69\% | 7/10 | 0.22\% | 1.39\% | 10/10 | 0.00\% | 0.00\% |
| 50 | 2/10 | 2.12\% | 6.11\% | 2/10 | 0.91\% | 2.60\% | 4/10 | 0.14\% | 0.54\% | 10/10 | 0.00\% | 0.00\% |
| Total/Avg. | 6/40 | 2.64\% | 6.54\% | 25/40 | 0.27\% | 0.83\% | 29/40 | 0.09\% | 0.49\% | 40/40 | 0.00\% | 0.00\% |

As the results show, BPC matched all previous best-known solutions and unveiled 11 new best-known solutions in all the replicates. On the subset of instances with 20 customers, both MSH and SLNS matched all the bestknown solutions. However, as the number of customers increases, the quality of the solutions found decreases.

Table 2 compares the performance of BPC in the best performing replicate against the benchmark algorithms in terms of the optimality gap. To compute the optimality gap we used the lower bound found by our BPC. Similar to Table 1 the results are grouped by instance size. Columns 2, 5, 8 , and 11 show the number of optimal solutions found by each algorithm. Columns 3, 6, 9 , and 12 contain the average optimality gap. Columns 4, 7, 10 , and 13 show the maximum optimality gap.

Table 2: Assessing optimality on the Coindreau et al. 2019) instances.

| $\|\mathcal{C}\|$ | VNS |  |  | MSH |  |  | SLNS |  |  |  |  | BPC |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | \#Opt. Avg. $\Delta$ | Max. $\Delta$ | \#Opt. Avg. $\Delta$ | Max. $\Delta$ | \#Opt. Avg. $\Delta$ | Max. $\Delta$ | \#Opt. Avg. $\Delta$ | Max. $\Delta$ |  |  |  |  |
| 20 | $1 / 10$ | $4.09 \%$ | $8.87 \%$ | $10 / 10$ | $0.00 \%$ | $0.00 \%$ | $10 / 10$ | $0.00 \%$ | $0.00 \%$ | $10 / 10$ | $0.00 \%$ | $0.00 \%$ |
| 30 | $0 / 10$ | $3.13 \%$ | $5.89 \%$ | $7 / 10$ | $0.01 \%$ | $0.04 \%$ | $8 / 10$ | $0.00 \%$ | $0.04 \%$ | $10 / 10$ | $0.00 \%$ | $0.00 \%$ |
| 40 | $3 / 10$ | $1.23 \%$ | $5.31 \%$ | $6 / 10$ | $0.15 \%$ | $0.69 \%$ | $7 / 10$ | $0.22 \%$ | $1.39 \%$ | $10 / 10$ | $0.00 \%$ | $0.00 \%$ |
| 50 | $2 / 10$ | $2.22 \%$ | $6.11 \%$ | $2 / 10$ | $1.00 \%$ | $2.60 \%$ | $4 / 10$ | $0.23 \%$ | $0.93 \%$ | $9 / 10$ | $0.09 \%$ | $0.91 \%$ |
| Total/Avg. | $6 / 40$ | $2.67 \%$ | $6.54 \%$ | $25 / 40$ | $0.29 \%$ | $0.83 \%$ | $29 / 40$ | $0.11 \%$ | $0.59 \%$ | $39 / 40$ | $0.02 \%$ | $0.23 \%$ |

Note that BPC is the first to prove optimality for 39 (out of 40 ) instances. Moreover, the average optimality gap of the solutions found by BPC is $0.02 \%$. Additionally, the maximum optimality gap on the unsolved instances is $0.91 \%$. With regard to the metaheuristics, SLNS has the best performance finding solutions with an average optimality gap of $0.11 \%$.

Finally, Table 3 compares the performance of each algorithm in terms of computational efficiency. Each row corresponds to an instance size. Columns $2,3,4$, and 5 show the average runtime in seconds reported by each algorithm. Columns 6 and 7 show the minimum and maximum CPU time employed by the BPC.

Table 3: Computational times on the Coindreau et al. 2019) instances.

| $\|\mathcal{C}\|$ | VNS | MSH | SLNS | BPC |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Avg. CPU (s) | Avg. CPU (s) | Avg. CPU (s) | Avg. CPU (s) | Min. CPU (s) | Max. CPU (s) |
| 20 | 71.0 | 13.4 | 15.0 | 28.1 | 22.1 | 40.1 |
| 30 | 381.0 | 35.9 | 30.0 | 75.9 | 51.3 | 127.7 |
| 40 | 1993.0 | 56.1 | 60.0 | 288.7 | 100.2 | 1340.8 |
| 50 | 6779.0 | 110.5 | 120.0 | 1877.4 | 276.6 | 7200.0 |
| Average | 2306.0 | 54.0 | 56.3 | 567.5 | 112.5 | 2177.1 |

While a perfect head-to-head comparison is hard to make because of differences in the programming languages and testing environment, the results suggest that in the subset of instances with 20 and 30 customers, BPC is close to match the performance of the state-of-the-art matheuristics. On average BPC uses less than 10 minutes.

### 5.3. Assessing the impact of the initialization step and the pruning strategies

As our proposed method uses a metaheuristic to initialize the set of columns, it is only logical to assess the performance of our BPC without this component. With this in mind, we ran our BPC while only using one iteration of MSH. This version of the algorithm is labeled as BPC-W. Moreover, to measure the impact of the problem-specific algorithmic components
that we designed, we ran our BPC with a version of the PA that does not include the path completion and the subtour fixing strategies. This version of our BPC is labeled as BPC-O.

Table 4 compares the performance of the branch-price-and-cut algorithms described above in terms of solution quality. Each row corresponds to a combination between a version of our BPC algorithm and an instance size. Recall that each algorithm was tested on every instance across five replicates in which the initial solution is modified. Column 3 shows the average number of best-known solutions found by each algorithm. Column 4 shows the average number of optimal solutions found by each algorithm. Columns 5, 6 , and 7 present the average, minimum, and maximum computational time in seconds used by each algorithm. Finally, columns 8, 9, and 10 show the average, minimum, and maximum number of columns.

Table 4: Solution quality of the BPC variants with $\zeta=5 \mathrm{~km}$.

| Algorithm | $\|\mathcal{C}\|$ | Avg. \#BKS | Avg. \# Optimal | CPU time (s) |  |  | \# Columns |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Avg. | Min | Max | Avg. | Min | Max |
| BPC | 20 | 10/10 | 10/10 | 28.11 | 22.12 | 40.13 | 219.30 | 85 | 461 |
|  | 30 | 10/10 | 10/10 | 75.90 | 51.29 | 127.67 | 393.74 | 92 | 693 |
|  | 40 | 10/10 | 10/10 | 288.72 | 100.17 | 1340.78 | 1647.88 | 387 | 6050 |
|  | 50 | 10/10 | 8.8/10 | 1877.42 | 276.60 | 7200.00 | 1864.00 | 779 | 5426 |
| BPC-O | 20 | 10/10 | 10/10 | 27.19 | 21.32 | 39.61 | 221.54 | 85 | 461 |
|  | 30 | 10/10 | 10/10 | 74.07 | 50.60 | 138.73 | 392.72 | 92 | 664 |
|  | 40 | 10/10 | 10/10 | 317.17 | 96.14 | 1755.69 | 1562.46 | 486 | 6824 |
|  | 50 | 9/10 | 8.2/10 | 2026.22 | 279.66 | 7200.00 | 1807.70 | 646 | 3934 |
| BPC-W | 20 | 10/10 | 10/10 | 14.43 | 4.15 | 41.62 | 2246.14 | 555 | 5155 |
|  | 30 | 10/10 | 10/10 | 118.84 | 28.60 | 342.65 | 12998.62 | 1586 | 35468 |
|  | 40 | 10/10 | 10/10 | 1018.09 | 242.95 | 4219.65 | 5871.14 | 1874 | 39067 |
|  | 50 | 8.8/10 | 8.4/10 | 2346.02 | 354.07 | 7200.00 | 10070.38 | 2312 | 53989 |

Note that BPC is the only algorithm that finds the best-known solution for every instance in all the replicates. Moreover, it is the algorithm that on average provides the highest number of optimal solutions. Nevertheless, note that BPC-W is capable of finding high quality solutions and of proving optimality of at least 38 out of 40 instances. This shows that the BPC solution quality is not exclusively due to the initialization procedure. In addition, note that BPC-O is not capable of finding one of the best-known solutions before reaching the running time limit, thus showing the importance of including problem-specific pruning strategies inside the PA.

With respect to the computational times, BPC is on average faster than its counterparts, especially on the subset of instances with 40 and 50 customers, where both the initialization procedure and the additional pruning
strategies improve the algorithm's performance. However, in the subset of instances with 20 customers, it seems that the overhead incurred by the initialization procedure does not pay off, as BPC-W is faster than BPC and delivers solutions of equivalent quality.

### 5.4. Analyzing the pricing problem algorithm

Most modern BPC algorithms for vehicle routing solve the pricing problem using labeling algorithms that rely on the NG-path relaxation proposed by (Baldacci et al., 2011). This relaxation consists in defining a neighborhood $\mathcal{N}_{i}$ for every customer $i \in \mathcal{C}$ that includes the X closest customers to $i$ and the customer itself. An NG-path can include cycles starting and ending at customer $j$ if and only if there exists a customer $i$ in the cycle such that $j \in \mathcal{N}_{i}$. Thus, such a cycle is forbidden if and only if $j \in \mathcal{N}_{i}$ for every customer $i$ it contains. The higher the value of X , the tighter the relaxation will be. Due to its success in column generation based algorithms, we believe it is interesting to compare the performance of our adaptation of the PA against a classical single-directional labeling algorithm implementing the NG-path relaxation.

With this objective, we solve the RSCP without applying any cuts or branching decisions. In the reminder of this section, the version of the algorithm using the PA to solve the pricing problem is labeled CG-PA and that using the labeling algorithm is labeled CG-NG-X. For these experiments we solve the instances with 20 and 30 customers under two settings. First, the case in which walking is not allowed, namely $\zeta=0 \mathrm{~km}$. Second, the case in which walking is allowed, namely $\zeta=5 \mathrm{~km}$. We consider these two settings to isolate the impact of including park-and-loop routes on both pricing algorithms.

Table 5 compares the performance of each column generation algorithm while solving the RSCP when walking is not allowed. Each row in the table corresponds to a pricing algorithm and a number of customers. Columns 3, 4 , and 5 show the average, minimum, and maximum lower bound produced by each approach. Columns 6,7 , and 8 , present the average, minimum, and maximum computational time in seconds. Finally, columns 9,10 , and 11 show the average, minimum, and maximum number of columns found by the corresponding pricing algorithm.

As the results show, CG-PA provides on average a better (higher) lower bound than the CG algorithms that use the NG-path relaxation. However, in this subset of instances the CG algorithms using the NG-path relaxation

Table 5: Performance of the CG algorithms solving the RSCP root node with $\zeta=0$.

| CG algorithm | $\|\mathcal{C}\|$ | Lower bound |  |  | CPU time (s) |  |  | \# Columns |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Avg. | Min | Max | Avg. | Min | Max | Avg. | Min | Max |
| CG-PA | 20 | 41.85 | 33.64 | 46.01 | 21.53 | 17.61 | 24.66 | 321.00 | 117 | 827 |
|  | 30 | 52.35 | 47.91 | 55.84 | 43.49 | 37.20 | 49.84 | 538.06 | 167 | 1046 |
| CG-NG-10 | 20 | 41.84 | 33.64 | 46.01 | 11.38 | 10.37 | 13.30 | 548.30 | 125 | 2592 |
|  | 30 | 52.22 | 47.78 | 55.84 | 21.72 | 18.58 | 26.35 | 769.04 | 215 | 1762 |
| CG-NG-7 | 20 | 41.75 | 33.43 | 46.01 | 11.53 | 10.34 | 15.38 | 758.88 | 171 | 3565 |
|  | 30 | 52.11 | 47.49 | 55.54 | 21.90 | 18.88 | 26.08 | 938.98 | 209 | 2364 |

solve the RSCP faster. With respect to the number of columns, CG-PA requires on average a lower number of columns. These results are expected, as both CG-NG-10 and CG-NG-7 are solving a relaxation of the pricing problem. Moreover, without the presence of walking subtours, the subtour fixing strategy used by the PA does not have an impact on the algorithm's performance.

Table 6 compares the performance of each column generation algorithm while solving the RSCP when walking is allowed. Similarly, each row in the table corresponds to a pricing algorithm and a number of customers. Columns 3, 4 , and 5 show the average, minimum, and maximum lower bound delivered by each approach. Columns 6,7 , and 8 , present the average, minimum, and maximum computational time in seconds. Finally, columns 9, 10, and 11 show the average, minimum, and maximum number of columns found by the corresponding pricing algorithm.

Table 6: Performance of the pricing algorithms solving the root node with $\zeta=5 \mathrm{~km}$.

| CG algorithm | $\|\mathcal{C}\|$ | Lower bound |  |  | CPU time (s) |  |  | \# Columns |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Avg. | Min | Max | Avg. | Min | Max | Avg. | Min | Max |
| CG-PA | 20 | 36.64 | 29.61 | 42.28 | 25.79 | 21.10 | 30.95 | 194.06 | 90 | 503 |
|  | 30 | 45.16 | 40.64 | 49.59 | 61.02 | 50.78 | 78.54 | 349.54 | 84 | 672 |
| CG-NG-10 | 20 | 36.62 | 29.61 | 42.28 | 35.94 | 21.36 | 69.54 | 287.70 | 95 | 666 |
|  | 30 | 45.08 | 40.21 | 49.59 | 370.70 | 127.06 | 918.51 | 475.64 | 72 | 1036 |
| CG-NG-7 | 20 | 36.51 | 28.90 | 42.28 | 32.81 | 19.23 | 63.07 | 349.28 | 89 | 641 |
|  | 30 | 44.89 | 40.00 | 49.59 | 273.68 | 94.67 | 749.04 | 591.86 | 78 | 1195 |

Note that the CG algorithms using the NG-paths relaxation provide lower bounds of lower quality in comparison with the bounds found by CG-PA. This can have a large impact on the solution algorithm as branch-and-bound nodes can be pruned early. Moreover, the computational time required to solve the RSCP is significantly larger for the NG-paths based algorithms.

Indeed, considering the subset of instances with 30 customers, on average CG-PA takes 61.02 seconds to solve the RSCP while CG-NG-10 and CG-NG7 take 370.70 and 270.68 seconds, respectively. A possible explanation for this behaviour is the soft dominance induced by the presence of the walking subtours, which affects the performance of the labeling algorithm. Finally, with respect to the number of columns CG-PA generates on average a smaller number of columns.

### 5.5. Assessing the importance of introducing park-and-loop routes

Introducing park-and-loop routes helps decreasing the driven distance. To assess their potential impact, we compare the objective function of the best-known solutions while varying the maximum walking distance. Namely, we consider the vehicle routing configuration (i.e., $\zeta=0$ ) and the park-andloop configuration with $\zeta$ equal to 5 or 10 kilometers.

Table 7 compares the average driven distance of each configuration. Each row corresponds to an instance size. Column 2 shows the average driven distance without allowing walking subtours. Column 3 and 5 show the average driven distance while imposing a limit of 5 or 10 kilometers on the total walking distance respectively. Finally, Column 4 and 6 show the average savings gained by introducing walking subtours.

Table 7: Average driven distance while varying the maximum walking distance.

| \# Customers | VRP |  |  | PLRP $(\zeta=5)$ |  |  | PLRP $(\zeta=10)$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Avg. km |  | Avg. km | $\% \Delta$ vs VRP |  | Avg. km | $\% \Delta$ vs VRP |  |
| 20 | 42.08 |  | 36.71 | $-12.8 \%$ |  | 32.63 | $-22.5 \%$ |  |
| 30 | 53.35 |  | 45.32 | $-15.0 \%$ |  | 38.04 | $-28.7 \%$ |  |
| 40 | 60.82 |  | 56.84 | $-6.5 \%$ |  | 54.97 | $-9.6 \%$ |  |
| 50 | 69.38 |  | 61.80 | $-10.9 \%$ |  | 60.43 | $-12.9 \%$ |  |
| Average | 56.41 |  | 50.17 | $\mathbf{- 1 1 . 3 \%}$ |  | 46.52 | $\mathbf{- 1 8 . 4 \%}$ |  |

As the results show, introducing walking subtours decreases the driven distance by $11.3 \%$ and $18.4 \%$ on average. As expected, increasing the walking distance limit allows for larger walking subtours and, in turn, for greater savings. Figure 3 shows the best-known solutions for instance $30 \_A_{-} 1$. When walking subtours are not allowed, the total driven distance is 48.81 km . If walking subtours are allowed, the driven distances are 40.73 km and 32.93 km while setting $\zeta$ to 5 and 10 kilometers, respectively. As stated before, major savings can be achieved by introducing park-and-loop routes. As this
figure shows, increasing the maximum walking distance has an impact on the size and length of the walking subtours present in each route.

We have developed a website available at https://chairelogistique. hec.ca/en/scientific-data//where researchers can download the instances, access instance-by-instance results, and upload their own solutions in order to encourage future research on this problem and make comparison with our results easier.


Figure 3: Solutions for instance 30_A_1.

## 6. Concluding remarks

In this paper, we presented a branch-price-and-cut algorithm for solving the PLRP. To do so, we formulated the PLRP as a set covering problem that considers a large set of paths. To solve this model, we use a column
generation approach that unveils promising paths that favor the objective of minimizing the total cost. The master problem selects paths to serve all customers, while the pricing problem generates feasible paths. We leveraged a tailored and improved version of the pulse algorithm to solve an elementary resource constrained shortest path with park-and-loop, that includes additional pruning strategies and allows for considering the subset row inequalities. Using this new version of the pulse algorithm improves the performance of the branch-price-and-cut algorithm.

We compared our branch-price-and-cut algorithm with the state-of-theart algorithms for solving a related problem called the vehicle routing problem with transportable resources without carpooling. The proposed algorithm was capable of finding all the previously best-known solutions. Moreover, it found 11 previously unknown optimal solutions. In addition, our method is the first capable of solving 39 instances to optimality out of the 40 instances that composed the testbed. Our experiments also show the advantages of providing a high quality pool of columns as a warm start as it significantly decreases the computational effort required by the algorithm.

We also showed the benefits of introducing a park-and-loop structure in the routing plan, as it allows decreasing the total driven distance. To encourage future research on the PLRP, we developed an online tool that allows the members of the community to visualize the best-known solutions and upload their own solutions for checking and plotting. Future research should focus on extending the branch-price-and-cut algorithm to solve the PLRP with time windows. Moreover, improving the performance of the pricing problem by implementing bidirectional search strategies and other problemspecific pruning strategies seems promising to decrease the computational time needed by each CG iteration.

## Acknowledgements

This research was partially funded by the Canada First Research Excellence Fund through IVADO and the Discovery grants. This research was also cofunded by HEC Montréal through the Chair in Logistics and Transportation and the Research Professorship on Clean Transportation Analytics.

## References

Arslan, O., Jabali, O., \& Laporte, G. (2018). Exact solution of the evasive flow capturing problem. Operations Research, 66, 1625-1640.

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

Belenguer, J., Benavent, E., Martinez, A., Prins, C., Prodhon, C., \& Villegas, J. (2016). A branch-and-cut algorithm for the single truck and trailer routing problem with satellite depots. Transportation Science, 50, 735749.

Cabrera, N., Cordeau, J.-F., \& Mendoza, J. E. (2022). The doubly open park-and-loop routing problem. Computers \& Operations Research, 143, 105761.

Cabrera, N., Medaglia, A. L., Lozano, L., \& Duque, D. (2020). An exact bidirectional pulse algorithm for the constrained shortest path. Networks, 76, 128-146.

Chao, I.-M. (2002). A tabu search method for the truck and trailer routing problem. Computers and Operations Research, 29, 33-51.

Coindreau, M., Gallay, O., \& Zufferey, N. (2019). Vehicle routing with transportable resources: Using carpooling and walking for on-site services. European Journal of Operational Research, 279, 996-1010.

Contardo, C., \& Martinelli, R. (2014). A new exact algorithm for the multidepot vehicle routing problem under capacity and route length constraints. Discrete Optimization, 12, 129-146.

Costa, L., Contardo, C., \& Desaulniers, G. (2019). Exact branch-price-andcut algorithms for vehicle routing. Transportation Science, 53, 946-985.

Derigs, U., Pullmann, M., \& Vogel, U. (2013). Truck and trailer routing problems, heuristics and computational experience. Computers and Operations Research, 40, 536-546.

Desaulniers, G., Desrosiers, J., \& Solomon, M. M. (2006). Column Generation volume 5. Springer, New York.

Jepsen, M., Petersen, B., Spoorendonk, S., \& Pisinger, D. (2008). Subsetrow inequalities applied to the vehicle-routing problem with time windows. Operations Research, 56, 497-511.

Le Colleter, T., Dumez, D., Lehuédé, F., \& Péton, O. (2023). Small and large neighborhood search for the park-and-loop routing problem with parking selection. European Journal of Operational Research. Forthcoming.

Lin, S., Yu, V., \& Chou, S. (2009). Solving the truck and trailer routing problem based on a simulated annealing heuristic. Computers and Operations Research, 36, 1683-1692.

Lozano, L., Duque, D., \& Medaglia, A. L. (2015). An exact algorithm for the elementary shortest path problem with resource constraints. Transportation Science, 50, 348-357.

Lozano, L., \& Smith, J. C. (2017). A backward sampling framework for interdiction problems with fortification. INFORMS Journal on Computing, 29, 123-139.

Lübbecke, M. E., \& Desrosiers, J. (2005). Selected Topics in Column Generation. Operations Research, 53, 1007-1023.

Marques, G., Sadykov, R., Deschamps, J.-C., \& Dupas, R. (2020). An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. Computers $\mathcal{E}$ Operations Research, 114, 104833.

Martinez-Sykora, A., McLeod, F., Lamas-Fernandez, C., Bektaş, T., Cherrett, T., \& Allen, J. (2020). Optimised solutions to the last-mile delivery problem in london using a combination of walking and driving. Annals of Operations Research, 295, 645-693.

Montoya, A., Guéret, C., Mendoza, J. E., \& Villegas, J. G. (2016). A multispace sampling heuristic for the green vehicle routing problem. Transportation Research Part C: Emerging Technologies, 70, 113-128.

Parragh, S., \& Cordeau, J. (2017). Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. Computers and Operations Research, 83, 28-44.

Pessoa, A., Sadykov, R., Uchoa, E., \& Vanderbeck, F. (2013). In-out separation and column generation stabilization by dual price smoothing. In V. Bonifaci, C. Demetrescu, \& A. Marchetti-Spaccamela (Eds.), Experimental Algorithms (pp. 354-365). Berlin, Heidelberg: Springer Berlin Heidelberg.

Pugliese, L. D. P., \& Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. Networks, 62, 183200.

Reed, S., Campbell, A. M., \& Thomas, B. W. (2022). Does parking matter? the impact of search time for parking on last-mile delivery optimization. arXiv:2107.06788.

Restrepo, M. I., Lozano, L., \& Medaglia, A. L. (2012). Constrained networkbased column generation for the multi-activity shift scheduling problem. International Journal of Production Economics, 140, 466-472.

Rothenbächer, A.-K., Drexl, M., \& Irnich, S. (2018). Branch-and-price-andcut for the truck-and-trailer routing problem with time windows. Transportation Science, 52, 1174-1190.

Schrotenboer, A. H., Ursavas, E., \& Vis, I. F. (2019). A branch-and-price-and-cut algorithm for resource-constrained pickup and delivery problems. Transportation Science, 53, 1001-1022.

Semet, F. (1995). A two-phase algorithm for the partial accessibility constrained vehicle routing problem. Annals of Operations Research, 61, 4565.

Sheuerer, S. (2006). A tabu search heuristic for the truck and trailer routing problem. Computers and Operations Research, 33, 894-909.

Villegas, J., Prins, C., Medaglia, A., \& Velasco, N. (2010). Grasp/vnd and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots. Engineering Applications of Artificial Intelligence, 23, 780-794.

Villegas, J., Prins, C., Medaglia, A., \& Velasco, N. (2013). A matheuristic for the truck and trailer routing problem. European Journal of Operational Research, 230, 231-244.

Villegas, J., Prins, C., Prodhon, C., Medaglia, A., \& Velasco, N. (2011). A grasp with evolutionary path relinking for the truck and trailer routing problem. Computers and Operations Research, 38, 1319-1334.

## Appendix A. Detailed results for each instance

Table A.8 shows the performance of each algorithm on each instance of the VRPTR without carpooling. Each row corresponds to an instance. Columns 2 to 5 show the objective function of the solution found by the VNS, MSH, SLNS, and the BPC algorithm respectively. Column 6 shows the objective function of the current best-known solution.

Table A.8: Detailed results on the Coindreau et al. (2019) instances.

| Instance | VNS | MSH | SLNS | BPC | BKS |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 20_A_1 | 32.4037 | 30.9482 | 30.9482 | 30.9482 | 30.9482 |
| 20_A_2 | 41.6349 | 41.5547 | 41.5547 | 41.5547 | 41.5547 |
| 20_A_3 | 39.4469 | 36.2344 | 36.2344 | 36.2344 | 36.2344 |
| 20_A_4 | 39.1459 | 36.0251 | 36.0251 | 36.0251 | 36.0251 |
| 20_A_5 | 35.2747 | 35.2747 | 35.2747 | 35.2747 | 35.2747 |
| 20_A_6 | 43.7627 | 42.2866 | 42.2866 | 42.2866 | 42.2866 |
| 20_A_7 | 40.1143 | 38.6881 | 38.6881 | 38.6881 | 38.6881 |
| 20_A_8 | 39.1395 | 36.8504 | 36.8504 | 36.8504 | 36.8504 |
| 20_A_9 | 29.9427 | 29.6105 | 29.6105 | 29.6105 | 29.6105 |
| 20_A_10 | 41.2404 | 39.6656 | 39.6656 | 39.6656 | 39.6656 |
| 30_A_1 | 41.2853 | 40.7372 | 40.7372 | 40.7372 | 40.7372 |
| 30_A_2 | 46.6485 | 45.6635 | 45.6465 | 45.6465 | 45.6465 |
| 30_A_3 | 50.9785 | 49.1828 | 49.1828 | 49.1828 | 49.1828 |
| 30_A_4 | 46.3310 | 43.7556 | 43.7556 | 43.7556 | 43.7556 |
| 30_A_5 | 48.3961 | 47.0619 | 47.0619 | 47.0619 | 47.0619 |
| 30_A_6 | 51.6505 | 49.5880 | 49.5880 | 49.5880 | 49.5880 |
| 30_A_7 | 45.5537 | 45.5537 | 45.5537 | 45.5340 | 45.5340 |
| 30_A_8 | 45.3691 | 43.6799 | 43.6799 | 43.6799 | 43.6799 |
| 30_A_9 | 41.8553 | 41.1629 | 41.1629 | 41.1609 | 41.1609 |
| 30_A_10 | 49.5241 | 46.8936 | 46.8936 | 46.8936 | 46.8936 |
| 40_A_1 | 59.1695 | 59.1695 | 59.1695 | 59.1695 | 59.1695 |
| 40_A_2 | 58.5186 | 58.5186 | 58.5186 | 58.1191 | 58.1191 |
| 40_A_3 | 63.1723 | 62.9222 | 62.9222 | 62.9222 | 62.9222 |
| 40_A_4 | 50.3730 | 50.3730 | 50.3730 | 50.3730 | 50.3730 |
| 40_A_5 | 52.7504 | 51.7278 | 51.7278 | 51.7278 | 51.7278 |
| 40_A_6 | 63.0658 | 61.2621 | 61.2621 | 61.2621 | 61.2621 |
| 40_A_7 | 57.7673 | 54.8824 | 55.6158 | 54.8554 | 54.8554 |
| 40_A_8 | 56.3737 | 56.3140 | 55.9774 | 55.9774 | 55.9774 |
| 40_A_9 | 56.4612 | 56.3871 | 56.3871 | 56.3013 | 56.3013 |
| 40_A_10 | 57.6954 | 57.6954 | 57.6954 | 57.6954 | 57.6954 |
| 50_A_1 | 60.5063 | 57.0235 | 57.0235 | 57.0235 | 57.0235 |
| 50_A_2 | 62.4825 | 62.1686 | 60.6031 | 60.5906 | 60.5906 |
| 50_A_3 | 65.8727 | 63.8855 | 63.6788 | 63.5529 | 63.5529 |
| 50_A_4 | 58.2333 | 57.9472 | 56.9391 | 56.6342 | 56.6342 |
| 50_A_5 | 64.1751 | 64.0909 | 64.0909 | 64.0909 | 64.0909 |
| 50_A_6 | 66.1254 | 65.0105 | 65.0105 | 64.8116 | 64.8116 |
| 50_A_7 | 63.6674 | 63.6674 | 63.6674 | 63.6350 | 63.6350 |
| 50_A_8 | 70.9057 | $69.7911^{34}$ | 68.8367 | 68.6323 | 68.6323 |
| 50_A_9 | 58.2212 | 58.7191 | 58.2212 | 58.2212 | 58.2212 |
| 50_A_10 | 60.7796 | 61.2239 | 60.7796 | 60.7796 | 60.7796 |


[^0]:    ${ }^{1}$ Available at https://chairelogistique.hec.ca/en/scientific-data/

[^1]:    ${ }^{2}$ The latest version of jORLib can be downloaded at: http://coin-or.github.io/jorlib/.

