The Workforce Scheduling and Routing Problem with Park-and-loop

Nicolás Cabrera[a], Jean-François Cordeau[a], Jorge E. Mendoza[a]

[a]*HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7 Canada*

Abstract

This paper introduces formulations and an exact algorithm for the workforce scheduling and routing problem with park-and-loop. This problem extends the standard workforce scheduling and routing problem by allowing the use of walking subtours in the routes. We introduce a compact arc-based formulation as well as a path-based formulation with an exponential number of variables. To efficiently solve the latter, we propose a branch-price-and-cut algorithm that leverages state-of-the-art techniques, including a tailored version of the pulse algorithm to solve the pricing problem and the separation of subset row inequalities to strengthen the lower bound. We report on computational experiments carried out on a set of instances with up to 75 tasks adapted from the literature. The results show that our method systematically outperforms a standard MIP solver, proving optimality for 232 out of 324 instances. We also report experiments on the closely-related service technician routing and scheduling problem, where our method delivered 12 new best solutions on a 54-instance testbed from the literature.

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

1. Introduction

In this paper, we study the workforce scheduling and routing problem with park-and-loop (WSRP-PL). While our initial inspiration came from a real-world application in France, similar problems are encountered by utility and service companies around the world. In our problem, a company must perform a set of on-site tasks (e.g., connection to utility grids, troubleshooting, meter reading). Each task has an associated duration and an associated

time window. In addition, depending on its nature, a task may require one or more skills, each at a potentially different level of proficiency. Tasks are executed either by the company's workforce or by a third party. Every worker in the force masters a subset of skills, each at a given proficiency level. Workers can work individually or in teams. To simplify the narrative, we assume that workers always work in teams (but a team can comprise just one worker). Teams depart from and must return to a single depot within working hours driving a vehicle (i.e., a car). Because many customers are located in densely populated areas, in which access to parking may be limited, workers can walk between nearby locations (after safely parking the car). The WSRP-PL consists in building a plan to execute all the tasks while minimizing the total operational cost. The latter is composed of (i) the outsourcing cost of tasks assigned to the third party and (ii) the cost of the total distance driven by the internal teams. A plan is defined by the assignment of workers to teams for the day and the routing of the vehicles driven by the teams.

Our problem is closely related to the park-and-loop routing problem (PLRP) and the workforce scheduling and routing problem (WSRP), both of which are NP-Hard. The PLRP is a variation of the vehicle routing problem (VRP), where routes consist of a primary tour completed using a vehicle, along with sub-tours performed on foot after parking the vehicle. In the PLRP, and most of its variants, the route duration and total walking distance are bounded. Coindreau et al. (2019) introduced the vehicle routing problem with transportable resources (VRPTR). In this variant of the PLRP, workers are allowed to share a vehicle (i.e., carpool). To address this problem, the authors proposed a variable neighborhood search (VNS) algorithm. They assessed their algorithm using a set of new instances containing up to 50 customers. The experiments showed that their VNS could provide solutions for all instances within a computing time of 10 hours. It is worth noting that, in contrast to our WSRP-PL, in the VRPTR, workers are considered identical and capable of fulfilling all tasks.

Cabrera et al. (2022) introduced the doubly open park-and-loop routing problem (DOPLRP). In this variant of the PLRP workers initiate and conclude their routes at customer locations. To address this problem, they proposed a customized implementation of the multi-space sampling heuristic (Mendoza & Villegas, 2013). They conducted experiments using a dataset of real-world instances provided by a French utility, featuring up to 3,000 customers. Their findings demonstrated that their method consistently produced solutions that resulted in significant cost savings compared to those

generated by the company's routing software. Additionally, they applied their method to instances introduced by Coindreau et al. (2019) and were able to enhance 32 out of the 40 previously best-known solutions. There are two key distinctions between our problem and their DOPLRP. First, in DOPLRP, tasks do not have time windows or skill requirements. Consequently, workers are not required to form teams. Second, in our WSRP-PL, a fundamental requirement is that all routes must both start and end at the depot.

In a similar vein, Le Colleter et al. (2023) conducted a study on the park-and-loop routing problem with parking selection (PLRP-PS). In this particular variant of the PLRP, the vehicle can only be parked at predefined parking locations. The authors introduced a small and large neighborhood search metaheuristic. To enhance the algorithm's efficiency, they implemented operators specifically designed for selecting parking spots. The algorithm unveiled eight new best-known solutions for the Coindreau et al. (2019) instances. These solutions were later improved or confirmed as optimal by Cabrera et al. (2023), who introduced a branch-price-and-cut (BPC) algorithm for the PLRP. Through their algorithm, they found optimal solutions for 39 out of the 40 Coindreau et al. (2019) instances. The key component of their method is the pulse algorithm (PA), employed for solving the pricing problem. They improved the classical PA with problem-specific pruning strategies that significantly accelerated the algorithm. It is worth noting that their BPC is not equipped to handle task skill requirements, team formation, or time windows.

The workforce scheduling and routing problem (WSRP) combines elements from both scheduling and routing problems. In the scheduling component of this problem, the objective is to assign workers (e.g., technicians, nurses, and security guards) to provide a service or complete tasks for customers. When making these worker assignments, various features are taken into account, including skills compatibility (Kovacs et al., 2012; Braekers et al., 2016; Chen et al., 2016), team formation (Bredström & Rönnqvist, 2008; Zamorano et al., 2018), multiple time periods (Tricoire et al., 2013; Guastaroba et al., 2021), and precedence constraints (Goel & Meisel, 2013; Pereira et al., 2020), among others. On the other hand, the routing component involves designing a set of routes that workers use to travel between different locations. This routing process takes into consideration factors such as customer time windows, the presence of multiple depots, and the potential use of alternative transportation modes. For further exploration of applica-

tions and variants of the WSRP, interested readers are referred to Castillo-Salazar et al. (2016) and Paraskevopoulos et al. (2017) for a comprehensive review.

Our problem is a generalization of the technician routing and scheduling problem (STRSP) introduced by Kovacs et al. (2012), one of the most studied WSRP variants. They solved two versions of the problem. The *no-team* version in which routes are carried out by individual workers, and the *team* version in which routes are designed for multiple workers. They proposed an adaptive large neighborhood search (ALNS) heuristic that includes a set of classical destroy and repair operators. Using this methodology, the authors provided high-quality solutions to instances with up to 100 tasks in less than two minutes. The authors also proposed a mathematical formulation that they solve by means of a mixed integer problem solver (i.e., CPLEX). They tested their exact method on instances with 25 tasks. As opposed to the WSRP-PL, in the STRSP workers are only allowed to drive between locations.

Most of the work on the no-team version of the STRSP has focused on heuristic algorithms (Xie et al., 2017; Zhou et al., 2020; Gu et al., 2022). Xie et al. (2017) proposed an iterated local search (ILS) algorithm that uses cleverly designed neighborhood structures. Their method was evaluated against the ALNS of Kovacs et al. (2012), showing an improved performance in both solution quality and speed. Building on the algorithm by Xie et al. (2017), Zhou et al. (2020) presented an iterated local search with hybrid neighborhood search (ILS-HNS) algorithm. This algorithm switches between small and large neighborhoods, which allows the algorithm to escape local optima. The proposed algorithm improved 12 of the best known solutions. Similarly, Gu et al. (2022) presented a Lagrangian iterated local search (L-ILS) algorithm which significantly outperforms ILS-HNS. Their study sets L-ILS as the state-of-the-art algorithm. Note, however, that neither of these methods is exact. Moreover, neither of these methods is capable of solving the team version of the STRSP.

Our study contributes to the existing literature in several key ways. First, we introduce the workforce scheduling and routing problem with park-and-loop, a problem that arises at the intersection of two challenging combinatorial problems: the PLRP and the WSRP. Both of these problems have gained increasing practical relevance due to concerns such as labor shortages and carbon emissions. Second, to tackle this challenging problem, we have developed an exact Branch-Price-and-Cut (BPC) algorithm that leverages state-of-the-

art techniques. Our computational experiments demonstrate the superior efficiency of our BPC algorithm when compared to a standard mixed-integer programming (MIP) solver. Third, we have applied our algorithm to provide optimality certificates for 24 instances on a standard testbed for the no-team version of the STRSP. Out of these, 12 represent new best-known solutions. Finally, we have created an online tool that enables researchers to visualize and download all the solutions and instances reported in our paper, enhancing accessibility and usability for the research community.

This paper is organized as follows. Section 2 formally introduces the WSRP-PL. Section 3 presents a path-based formulation for the problem. Section 4 describes the proposed branch-price-and-cut algorithm. Section 5 presents the computational experiments. Finally, Section 6 presents the conclusions and outlines potential paths for future research.

2. Problem description and arc-based formulation

The WSRP-PL can be formally defined on a complete and directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of directed arcs. The set of nodes comprises a start depot $\overline{0}$, an end depot $\underline{0}$, and the set of tasks $\mathcal{C} = \{1, ..., n\}$. Note that, $\overline{0}$ and $\underline{0}$ can represent the same or distinct geographical locations. Arcs in $\mathcal{A}$ represent the connections between two tasks or between a task and the depot. To perform the tasks, a set of workers $\mathcal{W} = \{1, ..., m\}$ are assigned to teams in the set $\mathcal{T}$. Note that $|\mathcal{T}|$ must be an upper bound on the maximum number of teams to form. A maximum of ω workers can be assigned to a team. Each team $t \in \mathcal{T}$ departs from and arrives to the depot after performing its route. Each route has a maximum duration ϕ. Teams can drive or walk between locations. Accordingly, each arc $(i, j) \in \mathcal{A}$ has four main attributes: the driving distance μ_{ij}, the driving time τ_{ij}, the walking distance δ_{ij}, and the walking time η_{ij}. The maximum distance that can be traveled on foot between two points is θ. Moreover, the maximum distance that can be traveled by a team on foot in one day is ζ. Driving the car involves a variable cost c^v per unit of distance while walking is assumed to be free of charge.

Each task $i \in \mathcal{C}$ has a duration s_i and an associated time window indicating possible visit times. Let $[a_i, b_i]$ be the earliest and latest starting time of task $i \in \mathcal{C}$. Also, let f_i be the outsourcing cost of task $i \in \mathcal{C}$. Skill requirements are represented by ν_{iql}, an integer parameter stating the number of workers with the skill $q \in \mathcal{Q}$ with at least a proficiency level $l \in \mathcal{L}$ that

the task $i \in \mathcal{C}$ needs. Worker qualifications are represented by ξ_{kql}, which is a binary parameter equal to 1 if worker k has at least a proficiency level l for skill q. The objective of the WSRP-PL is to minimize the total cost while ensuring that: each task is fulfilled precisely once; the total duration of each route does not exceed the working day duration; and the total walking distance of each team does not exceed the distance limit. Figure 1 shows an example of a feasible solution to a toy WSRP-PL instance with nine tasks. In this example, there are four workers (1 to 5) and four potential skills (blue, yellow, green, and red), each with two proficiency levels (dark and light tones). To complete the tasks, the workers are divided into two teams. The green team is formed by workers 1 and 2 while the blue team is formed by workers 4 and 5. Worker 3 is not assigned to a team and remains at the depot for the day. The green team completes tasks 1, 2, and 4; the blue team completes tasks 5 to 9; and task 3 is outsourced.

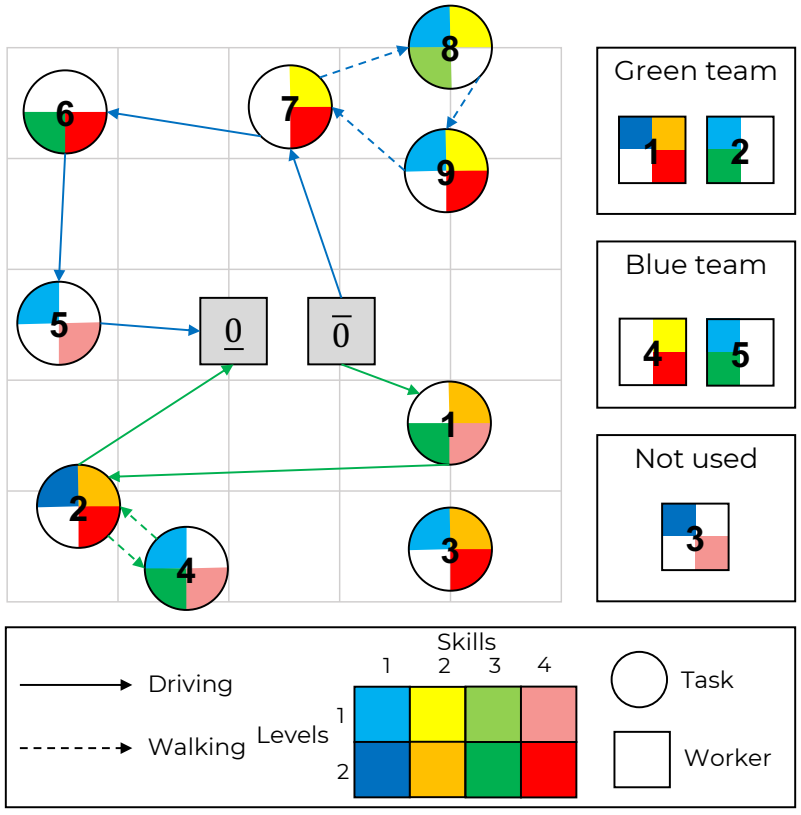

Figure 1: WSRP-PL solution example.

To mathematically state the WSRP-PL, we start by defining an additional set $\mathcal{N}^+ = \{n+1, n+2, \ldots, 2n\} \cup \left\{\overline{0}^*\right\}$ that contains a copy of every node $i \in \mathcal{N} \backslash \{\underline{0}\}$. Any node $(i+n) \in \mathcal{N}^+$ represents the end of a subtour starting at node $i \in \mathcal{C}$. Node $\overline{0}^*$ represents the end of a subtour starting at $\overline{0}$. Let $d(i)$ be the copy of node $i \in \mathcal{N} \setminus \{\underline{0}\}$. We also define two sets of arcs. Arc set $\mathcal{A}^1 = \{(i,j) : i \in (\mathcal{N} \cup \mathcal{N}^+) \setminus \{\underline{0}\}, j \in \mathcal{N}\}$ contains the arcs in which teams can drive. Arc set $\mathcal{A}^2 = \left\{(i,j) : i \in \mathcal{N} \setminus \{\underline{0}\}, j \in (\mathcal{N} \cup \mathcal{N}^+) \setminus \left\{\overline{0}, \underline{0}\right\} | \delta_{ij} \leq \theta\right\}$ contains the arcs in which teams can walk. The reader should note that some arcs belong to both $\mathcal{A}^1$ and $\mathcal{A}^2$. Building on top of the formulations presented by Kovacs et al. (2012) and Cabrera et al. (2022), the arc-based integer programming formulation (AF) of the WSRP-PL uses the following binary variables:

- $x_{ij}^t = 1$ if team $t \in \mathcal{T}$ drives from node i to node j, for $(i,j) \in \mathcal{A}^1$ and 0, otherwise,
- $h_{ij}^t = 1$ if team $t \in \mathcal{T}$ walks from node i to node j, for $(i,j) \in \mathcal{A}^2$ and 0, otherwise,
- $g_i^t = 1$ if team $t \in \mathcal{T}$ parks the vehicle at node $i \in \mathcal{N} \setminus \{\underline{0}\}$ and 0, otherwise,
- $y_i^t = 1$ if team $t \in \mathcal{T}$ is assigned to complete task $i \in \mathcal{C}$ and 0, otherwise,
- $v_k^t = 1$ if worker $k \in \mathcal{W}$ is assigned to team $t \in \mathcal{T}$ and 0, otherwise,
- $z_i = 1$ if task $i \in \mathcal{C}$ is outsourced and 0, otherwise,
- $w_t = 1$ if team $t \in \mathcal{T}$ is selected and 0, otherwise.

We also define the following continuous variables:

- u_i^t is the arrival time of team $t \in \mathcal{T}$ to node $i \in \mathcal{N} \cup \mathcal{N}^+$.

An arc-based formulation for the WSRP-PL can be stated as follows:

$$\min \sum_{(i,j) \in \mathcal{A}^1} \sum_{t \in \mathcal{T}} x_{ij}^t \mu_{ij} c^v + \sum_{i \in \mathcal{C}} z_i f_i \tag{1}$$

subject to

$$\sum_{t\in\mathcal{T}} v_k^t \leq 1 \qquad \forall k \in \mathcal{W} \tag{2}$$

$$\sum_{k\in\mathcal{W}} v_k^t \geq w_t \qquad \forall t \in \mathcal{T} \tag{3}$$

$$\sum_{k\in\mathcal{W}} v_k^t \leq w_t\omega \qquad \forall t \in \mathcal{T} \tag{4}$$

$$w_t \geq w_{t+1} \qquad \forall t \in \mathcal{T} | t < |\mathcal{T}| \tag{5}$$

$$w_t - \sum_{i\in\mathcal{C}} y_i^t \leq 0 \qquad \forall t \in \mathcal{T} \tag{6}$$

$$\sum_{t\in\mathcal{T}} y_i^t + f_i = 1 \qquad \forall i \in \mathcal{C} \tag{7}$$

$$\sum_{(\bar{0},j)\in\mathcal{A}^1} x_{\bar{0}j}^t + \sum_{(\bar{0},j)\in\mathcal{A}^2} h_{\bar{0}j}^t = 1 \qquad \forall t \in \mathcal{T} \tag{8}$$

$$\sum_{(i,\underline{0})\in\mathcal{A}^1} x_{i\underline{0}}^t = 1 \qquad \forall t \in \mathcal{T} \tag{9}$$

$$\sum_{(i,j)\in\mathcal{A}^1} x_{ij}^t + \sum_{(i,j)\in\mathcal{A}^2} h_{ij}^t = y_j^t \qquad \forall j \in \mathcal{C}, t \in \mathcal{T} \tag{10}$$

$$\sum_{(j,d(i))\in\mathcal{A}^2} h_{jd(i)}^t = g_i^t \qquad \forall i \in \mathcal{N} \setminus \{\underline{0}\}, t \in \mathcal{T} \tag{11}$$

$$\sum_{(i,j)\in\mathcal{A}^2} h_{ij}^t - \sum_{(j,i)\in\mathcal{A}^2} h_{ji}^t = g_i^t \qquad \forall i \in \mathcal{N} \setminus \{\underline{0}\}, t \in \mathcal{T} \tag{12}$$

$$\sum_{(j,i)\in\mathcal{A}^2} h_{ji}^t + \sum_{(j,i)\in\mathcal{A}^1} x_{ji}^t - \sum_{(i,j)\in\mathcal{A}^2} h_{ij}^t - \sum_{(i,j)\in\mathcal{A}^1} x_{ij}^t = 0 \qquad \forall j \in \mathcal{N} \cup \mathcal{N}^+, t \in \mathcal{T} \tag{13}$$

$$\sum_{(j,i)\in\mathcal{A}^2} h_{ji}^t + \sum_{(j,i)\in\mathcal{A}^1} x_{ji}^t \leq 1 \qquad \forall j \in \mathcal{N} \cup \mathcal{N}^+, t \in \mathcal{T} \tag{14}$$

$$\sum_{t\in\mathcal{T}} h_{ij}^t + x_{ij}^t \leq 1 \qquad \forall (i,j) \in \mathcal{A}^1 \cap \mathcal{A}^2 \tag{15}$$

$$x_{ij}^t\tau_{ij} + h_{ij}^t\eta_{ij} + y_i^t s_i + u_i^t - \phi(1 - h_{ij}^t - x_{ij}^t) \leq u_j^t \qquad \forall (i,j) \in \mathcal{A}^1 \cap \mathcal{A}^2, t \in \mathcal{T} \tag{16}$$

$$x_{ij}^t \tau_{ij} + y_i^t s_i + u_i^t - \phi(1 - x_{ij}^t) \leq u_j^t \qquad \forall (i,j) \in \mathcal{A}^1 \setminus (\mathcal{A}^1 \cap \mathcal{A}^2), t \in \mathcal{T} \tag{17}$$

$$h_{ij}^t \eta_{ij} + y_i^t s_i + u_i^t - \phi(1 - h_{ij}^t) \leq u_j^t \qquad \forall (i,j) \in \mathcal{A}^2 \setminus (\mathcal{A}^1 \cap \mathcal{A}^2), t \in \mathcal{T} \tag{18}$$

$$u_i^t \leq u_{d(i)}^t \qquad \forall i \in \mathcal{N} \setminus \{\underline{0}\}, t \in \mathcal{T} \tag{19}$$

$$u_i^t \leq b_i \qquad \forall i \in \mathcal{C}, t \in \mathcal{T} \tag{20}$$

$$u_i^t \geq a_i \qquad \forall i \in \mathcal{C}, t \in \mathcal{T} \tag{21}$$

$$\sum_{(i,j) \in \mathcal{A}^2} h_{ij}^t \delta_{ij} \leq \zeta \qquad \forall t \in \mathcal{T} \tag{22}$$

$$y_i^t \xi_{iql} - \sum_{k \in \mathcal{W}} v_k^t \nu_{kql} \leq 0 \qquad \forall t \in \mathcal{T}, q \in \mathcal{Q}, l \in \mathcal{L}, i \in \mathcal{C} \tag{23}$$

$$x_{ij}^t \in \{0,1\} \qquad \forall t \in \mathcal{T}, (i,j) \in \mathcal{A}^1 \tag{24}$$

$$h_{ij}^t \in \{0,1\} \qquad \forall t \in \mathcal{T}, (i,j) \in \mathcal{A}^2 \tag{25}$$

$$g_i^t \in \{0,1\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{N} \setminus \{\underline{0}\} \tag{26}$$

$$y_i^t \in \{0,1\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{C} \tag{27}$$

$$v_k^t \in \{0,1\} \qquad \forall t \in \mathcal{T}, k \in \mathcal{W} \tag{28}$$

$$z_i \in \{0,1\} \qquad \forall i \in \mathcal{C} \tag{29}$$

$$w_t \in \{0,1\} \qquad \forall t \in \mathcal{T} \tag{30}$$

$$u_i^t \geq 0 \qquad \forall t \in \mathcal{T}, i \in \mathcal{N} \cup \mathcal{N}^+. \tag{31}$$

The objective function (1) minimizes the total cost comprised of the routing and outsourcing costs. Constraints (2) ensure that a worker is only assigned to one team. Constraints (3) and (4) impose a minimum and a maximum number of workers for each selected team. Constraints (5) remove symmetric solutions with respect to the selection of teams. Constraints (6) state that every selected team must complete at least one task. Constraints (7) ensure that all tasks are either completed or outsourced. Constraints (8) state that all teams must leave the start depot. Constraints (9) ensure that all the teams arrive at the end depot.

Constraints (10) guarantee that all the tasks are executed by a team. Constraints (11) ensure that the vehicle is recovered after performing a subtour. Constraints (12) and (13) ensure flow conservation. Constraints (14)

impose a limit on the number of arcs that can be used by a team departing from any location. Constraints (15) state that an arc can only be used once. Constraints (16)-(19) define the time of arrival at every location. Constraints (20)-(21) ensure that the completion of each task starts within their time windows. Constraints (22) impose a limit on the duration the overall walking distance. Constraints (23) ensure that tasks are fulfilled by a team with the appropriate skills.

The proposed arc-based formulation is compact (i.e., the number of variables and constraints is polynomial with respect to the dimension of a problem instance). However, as mentioned by Dellaert et al. (2019), arc-based formulations suffer from poor performance caused by the bad quality of the lower bound obtained by solving its LP relaxation. As a result, we propose using a new path-based formulation which is the topic of the next section.

3. Path-based formulation

Let $\bar{\mathcal{G}} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$ be a directed graph, henceforth referred to as the *modified network*, where $\bar{\mathcal{N}}$ is the set of nodes and $\bar{\mathcal{A}}$ is the set of directed arcs. The set of nodes $\bar{\mathcal{N}} = \mathcal{N} \cup \{\mathfrak{s}\} \cup \mathcal{W}$ comprises the nodes in graph $\mathcal{G}$, a source node $\mathfrak{s}$, and one node for every worker $k \in \mathcal{W}$. There are two types of arcs in the modified network, namely, routing and scheduling arcs. Routing arcs in $\mathcal{A}$ represent connections between tasks, as defined in Section 2. Scheduling arcs represent the selection of workers. More precisely, let $\mathcal{A}' = \mathcal{A}'_1 \cup \mathcal{A}'_2 \cup \mathcal{A}'_3$ be the scheduling arcs, where $\mathcal{A}'_1 = \{(\mathfrak{s}, k) : k \in \mathcal{W}\}$ are arcs from the source node to every worker node, $\mathcal{A}'_2 = \left\{(k, \bar{0}) : k \in \mathcal{W}\right\}$ are arcs from every worker node to the start depot node, and $\mathcal{A}'_3 = \{(k, k') : k, k' \in \mathcal{W} | k' > k\}$ are arcs between workers nodes. A worker $k \in \mathcal{W}$ is said to be selected if an arc ending at the corresponding node is used. Figure 2 shows an illustrative example of the modified network on an instance that includes five workers and four tasks. The maximum number of workers in a team is three. The number of skills is set to four and each skill has two proficiency levels. For the sake of simplicity, some routing arcs were omitted. Note that by using this graph, both scheduling and routing decisions can be made simultaneously.

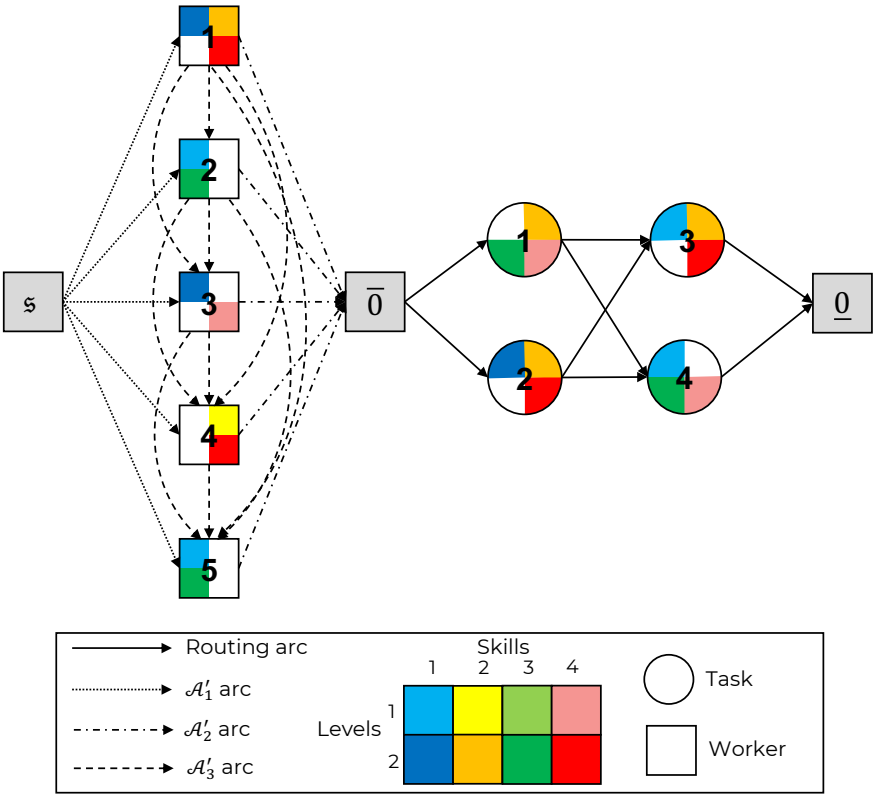

Figure 2: Modified network example.

Let path p be an ordered set of directed arcs starting at the source node $\mathfrak{s}$ and ending at the end depot $\underline{0}$. Let also $\mathcal{P}$ be the set of all feasible paths. The scheduling arcs in p are contained in subsets $\mathcal{A}_p^s$. The driving and the walking arcs in p are contained in subsets $\mathcal{A}_p^d$ and $\mathcal{A}_p^w$, respectively. Let $\mathcal{W}_p$ be the workers selected in the path. Also, let $\mathcal{C}_p$ be the tasks completed in the path starting the service within their corresponding time windows. A path p is feasible if the following conditions hold:

$$\sum_{(i,j)\in\mathcal{A}_p^s} 1 \leq \omega + 1 \tag{32}$$

$$\sum_{(i,j)\in\mathcal{A}_p^w} \eta_{ij} + \sum_{(i,j)\in\mathcal{A}_p^d} \tau_{ij} + \sum_{i\in\mathcal{C}_p} s_i \leq \phi \tag{33}$$

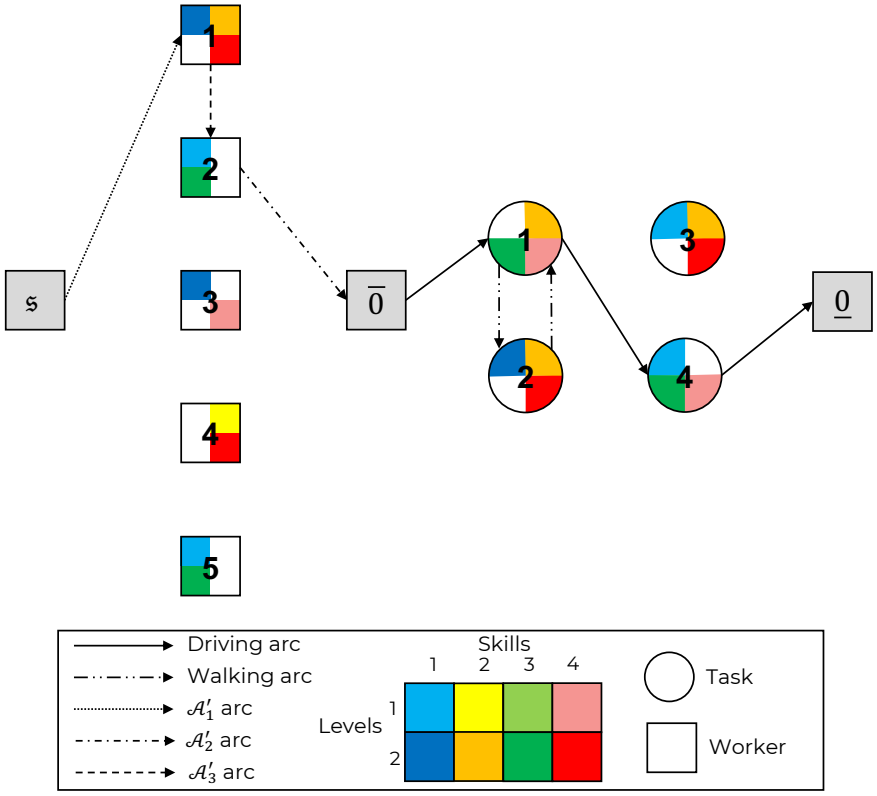

Figure 3: Feasible path example.

$$\sum_{(i,j)\in\mathcal{A}_p^w} \delta_{ij} \leq \zeta \tag{34}$$

$$\sum_{k\in\mathcal{W}_p} \nu_{kql} \geq \xi_{iql} \qquad \forall i \in \mathcal{C}_p, q \in \mathcal{Q}, l \in \mathcal{L}. \tag{35}$$

Condition (32) ensures that the maximum number of workers per team is respected. Condition (33) states that the total duration of a path must be less than the time limit. Condition (34) guarantees that the total walking distance respects the limit. Conditions (35) state that all tasks completed in the path must be fulfilled by a team with the appropriate skills. Figure 3 shows an example of a path carried out by a team composed of workers 1 and 2. This team fulfills tasks 1, 2, and 4 (in that order).

The reader should note that the path shown in Figure 3 could alternatively use worker 5 instead of worker 2. More generally, consider the case

in which two workers k and k' have the same qualifications $\xi_{kql} = \xi_{k'ql}$ for every skill $q \in \mathcal{Q}$ and proficiency level $l \in \mathcal{L}$. Then, every path $p \in \mathcal{P}$ that uses worker k can be replicated by using worker k'. To deal with this potential source of inefficiency due to symmetry, we slightly modify graph $\bar{\mathcal{G}}$ as follows. First, we define a set of worker profiles $\mathcal{O}$. All the workers with the same skills qualifications are associated with one unique worker profile. Let Γ_o be the number of available workers with profile $o \in \mathcal{O}$. Then, we can re-define node set $\bar{\mathcal{N}}$ as $\bar{\mathcal{N}} = \mathcal{N} \cup \{\mathfrak{s}\} \cup \mathcal{O}$, in which instead of having one node for every worker, the graph has one node per worker profile. Also, we re-define the set of scheduling arcs $\mathcal{A}'$ as $\mathcal{A}' = \mathcal{A}'_1 \cup \mathcal{A}'_2 \cup \mathcal{A}'_3$, where $\mathcal{A}'_1 = \{(\mathfrak{s}, o)^\alpha : o \in \mathcal{O}, \alpha \in 1 \dots \Gamma_o | \alpha \leq \omega\}$ are arcs from the source to every worker profile node, $\mathcal{A}'_2 = \left\{(o, \overline{0}) : o \in \mathcal{O}\right\}$ are arcs from every profile node to the start depot node, and $\mathcal{A}'_3 = \{(o, o')^\alpha : o, o' \in \mathcal{O}, \alpha \in 1 \dots \Gamma_{o'} | o' > o \wedge \alpha \leq \omega\}$. A worker profile $o \in \mathcal{O}$ is said to be selected if an arc $(\cdot, o)^\alpha$ ending at the corresponding node is used. The number of workers used of the associated worker profile is α.

Figure 4 shows how the graph presented in Figure 2 can be represented using the updated modified network. Numbers next to the arcs represent the number of workers selected if the arc is used. By convention, if no number is shown along the arc, the number of workers is 1. Note that workers 2 and 5 are now represented by a single node. Moreover, note that there are two arcs from the source node to the node corresponding to profile 2. Similarly, there are two arcs between the nodes representing profile 1 and profile 2. As a result, this alternative representation allows us to remove redundant paths from the graph.

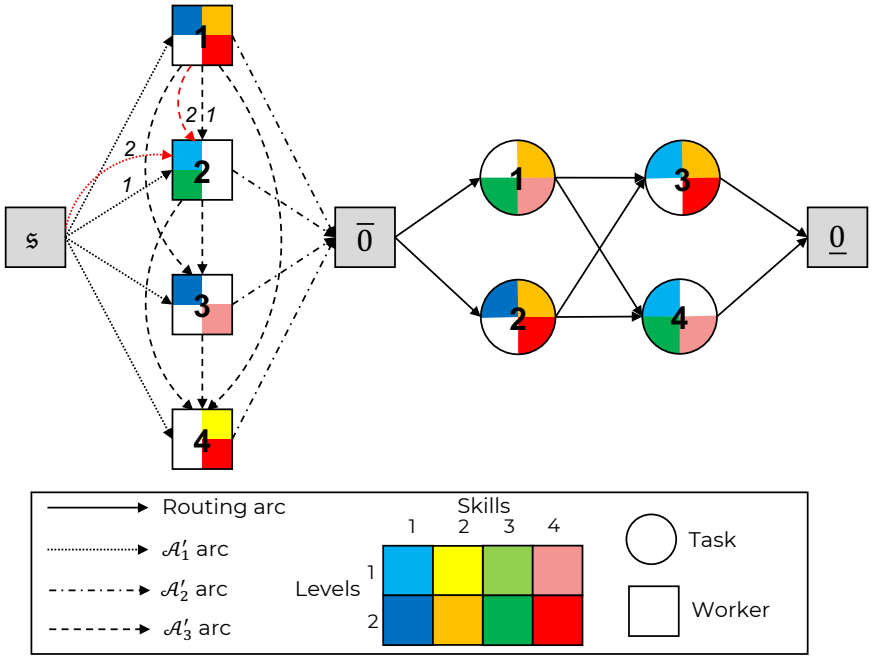

Figure 4: Modified network example using worker profiles.

The cost c_p of a path is equal to the driving cost, that is,

$$c_p = \sum_{(i,j)\in\mathcal{A}_p^d} \mu_{ij} c^v. \tag{36}$$

Let a_{ip} be a binary parameter that takes the value 1 if and only if path $p \in \mathcal{P}$ completes task $i \in \mathcal{C}$, and let b_{op} be a parameter that takes the value of the number of workers of profile $o \in \mathcal{O}$ in the team assigned to path $p \in \mathcal{P}$. Also, let ϑ_i be a binary variable equal to 1 if task $i \in \mathcal{C}$ is outsourced and 0 otherwise. Finally, let λ_p be a binary variable equal to 1 if path $p \in \mathcal{P}$ is selected and 0 otherwise. A set partitioning (SP) formulation for the WSRP-PL can be stated as follows:

$$\min \sum_{p\in\mathcal{P}} \lambda_p c_p + \sum_{i\in\mathcal{C}} \vartheta_i f_i \tag{37}$$

subject to

$$\sum_{p\in\mathcal{P}} a_{ip}\lambda_p + \vartheta_i = 1 \qquad \forall i \in \mathcal{C} \tag{38}$$

$$\sum_{p \in \mathcal{P}} b_{op} \lambda_p \leq \Gamma_o \qquad \forall o \in \mathcal{O} \tag{39}$$

$$\lambda_p \in \{0,1\} \qquad \forall p \in \mathcal{P} \tag{40}$$

$$\vartheta_i \in \{0,1\} \qquad \forall i \in \mathcal{C}. \tag{41}$$

The objective function (37) minimizes the total cost. Constraints (38) ensure that all tasks are fulfilled or outsourced. Constraints (39) guarantee that the number of workers used of each profile is less or equal than the number available. The SP model has one variable for every possible feasible path. However, the number of feasible paths $|\mathcal{P}|$ grows exponentially with the number of tasks and workers. As a result, enumerating all the feasible paths to solve SP is only possible for trivial instances. As an alternative, the SP can be solved using a branch-price-and-cut algorithm, which is described next.

4. Solution method

We propose a BPC algorithm for the above SP formulation. BPC is a branch-and-bound procedure for solving mixed integer linear programming models with a large number of variables. The algorithm's objective is to find the optimal solution without explicitly enumerating the complete set of feasible variables (paths). Instead, the algorithm iteratively builds a subset $\overline{\mathcal{P}} \subseteq \mathcal{P}$ of promising paths. To do so, the algorithm relies on two optimization problems. A *master problem* that evaluates the performance of the current subset of paths and a *pricing problem* that finds new promising paths. Usually, the subset $\overline{\mathcal{P}}$ is initialized using heuristic methods. We refer the interested reader to Costa et al. (2019) for a review of techniques and applications of BPC algorithms.

In our case, the master problem is the relaxed version of formulation (37)-(41). The master problem is obtained by relaxing the integrality constraints on the λ_p and ϑ_i variables. Let $\pi_i \in \mathbb{R}$ and $\sigma_o \leq 0$ be the dual variables associated with constraints (38) and (39), respectively. Then, the pricing problem, which aims to find paths with the most negative reduced cost is formulated as follows:

$$\min_{p \in \mathcal{P}} \left\{ \mathfrak{r}_p = c_p - \sum_{i \in \mathcal{C}} a_{ip} \pi_i - \sum_{o \in \mathcal{O}} b_{op} \sigma_o \right\}. \tag{42}$$

If the pricing problem finds a path with $\mathfrak{r}_p < 0$, we add the corresponding path p to the subset $\overline{\mathcal{P}}$. In the opposite case, the current master problem is solved optimally. If the optimal solution to the master problem is fractional, the algorithm uses a *cut separator* to generate valid inequalities and strengthen the linear relaxation. Lastly, if the solution is still fractional, the algorithm resorts to branching. In the following subsections, we present a detailed description of the algorithm used to solve the pricing problem, the initialization procedure, the cut separator, and the branching strategy.

4.1. Pricing problem

The pricing problem is an elementary shortest path problem with resource constraints (ESPPRC) from the source node $\mathfrak{s}$ to the end depot $\underline{0}$. A distinctive feature of our pricing problem is that paths have a park-and-loop structure. Also, paths are constrained by three resources: the time ϕ, the walking distance ζ, and the size of the team ω. To solve the pricing problem we use the pulse algorithm (PA), which is a modification of the algorithm originally proposed by Lozano et al. (2016) and later extended by Cabrera et al. (2023) to handle the park-and-loop structure of the paths. The PA is a recursive algorithm that performs a depth-first search exploration of the network by propagating pulses (i.e., partial paths) from the source node $\mathfrak{s}$. To avoid enumerating all the possible paths in the graph, the algorithm uses pruning strategies that aggressively and effectively discard partial paths. In particular, we use bounds, infeasibility, and rollback pruning strategies. Following the ideas of Cabrera et al. (2020) we do not solve the pricing problem to optimality at every iteration. To achieve this, we allow the PA to heuristically terminate the search if one of the following two conditions holds:

- The PA has found at least Υ paths with negative reduced cost;
- The PA has found at least one path with negative reduced cost and the computational time spent during the search is higher than Λ.

In our particular case, we improve the PA's performance by taking advantage of the structure of the graph $\bar{\mathcal{G}}$. In the original PA, a one-time quick pre-processing procedure is used to find conditional lower bounds $\underline{r}(i,t)$ on the reduced cost that can be achieved from every node $i \in \mathcal{C}$ to the end depot $\underline{0}$ with a given amount of consumed resource $t < \phi$. In a one-to-one implementation of the PA, this would lead to weak bounds, as information regarding the team configuration would be ignored. Thus, we adapted the PA to run

the pre-processing procedure every time the start depot $\overline{0}$ is reached by a partial path $\mathcal{P}$, using as an input the selected workers $\mathcal{W}_{\mathcal{P}}$. This modification has two main advantages. First, task nodes can be temporarily removed from graph $\bar{\mathcal{G}}$, which significantly quickens the pre-processing procedure and the pulse propagation. Second, the lower bounds found during the pre-processing procedure are stronger, which increases the chance of pruning a partial path earlier using the bounds pruning strategy. In addition, following the ideas of Lozano & Medaglia (2013), our pulse algorithm propagates different partial paths in parallel. In practice, we trigger multiple threads at the source node $\mathfrak{s}$ that begin to propagate partial paths starting from different outgoing arcs. This allows the algorithm to test different team configurations almost simultaneously. Similarly, once a partial path reaches the start depot $\overline{0}$, it also triggers multiple threads that propagate partial paths using the current team configuration.

4.2. Initial set of paths

The master problem is always feasible as it is always possible to outsource all tasks. Thus, it is entirely possible to start the BPC algorithm with an empty set of paths. However, it has been observed by multiple researchers that the performance of BPC algorithms is largely affected by the initial set of variables. Our algorithm follows a two-step approach to generate paths carried out by single-worker teams. First, we run our own implementation of the constructive heuristic proposed by Xie et al. (2017). Second, we apply a modified version of the tabu search metaheuristic proposed by Lozano et al. (2016) that considers the skills required by each task to quickly improve those paths.

4.3. Cut separator

The optimal solution to the master problem can be fractional. In that situation, the cut separator tries to generate valid subset row inequalities (cuts). These cuts were proposed by Jepsen et al. (2008) and have since then become essential in BPC algorithms for solving many vehicle routing problems (Contardo & Martinelli, 2014; Costa et al., 2019; Marques et al., 2020). The subset row inequalities with $|\mathcal{S}| = 3$ can be defined as follows:

$$\sum_{p \in \mathcal{P}} \left\lfloor 1/2 \sum_{i \in \mathcal{S}} a_{ip} \right\rfloor \lambda_p \leq 1, \qquad \forall \mathcal{S} \subseteq \mathcal{C}. \tag{43}$$

These inequalities guarantee that for every given subset $\mathcal{S} \subseteq \mathcal{C}$ of tasks, the number of paths that fulfill two or more of them is less than or equal to 1. Adding these inequalities to the master problem changes the formulation of the pricing problem. Let $\mathcal{S}$ be the subset of triplets of customers for which the subset row inequality has been generated and added to the master problem. Also, let $\beta_{\mathcal{S}} \leq 0$ be the dual variable associated with subset $\mathcal{S}$. Then, the pricing problem is formulated as:

$$\min_{p \in \mathcal{P}} \left\{ \mathfrak{r}_p = c_p - \sum_{i \in \mathcal{C}} a_{ip} \pi_i - \sum_{o \in \mathcal{O}} b_{op} \sigma_o - \sum_{\mathcal{S} \in \mathcal{S}} \beta_{\mathcal{S}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_{ip} \right\rfloor \right\}. \tag{44}$$

This formulation is usually more challenging to solve, particularly if the number of subset row inequalities already added to the master problem is high. As a consequence, the cut separator follows three steps. First, it enumerates all task triplets and checks if the current fractional solution violates the associated inequality by at least ε units. If so, the inequality is added to a list. After this step is completed, the list is sorted ensuring that inequalities with greater violations remain on top. Finally, the cut separator adds the first φ inequalities to the master problem. We set φ to 10 and ε to 0.1.

The PA used to solve the pricing problem handles these inequalities by adding a new resource for each subset $\mathcal{S} \in \mathcal{S}$. These resources keep track of the number of tasks in the subset that have been fulfilled. Every time that one of these resources reaches a value of 2, the associated dual variable $\beta_{\mathcal{S}}$ is subtracted from the objective function.

4.4. Branching strategy

Even after adding inequalities, the optimal solution of the master problem can remain fractional. In such a case, the algorithm uses a set of branching rules. We implement a five-stage hierarchical branching. At all levels, we branch on the variable for which the fractional value is closer to 0.5. The first level branches on driving flow variables. We branch on the implicit variable x_{ij} which equals 1 if any team drives through arc $(i, j) \in \mathcal{A}$, and 0 otherwise. To enforce $x_{ij} = 0$ we remove all the paths that involve driving between tasks i and j from the master problem and we also forbid the PA to use the arc while solving the pricing problem. To enforce $x_{ij} = 1$ we remove all the paths in the master problem that fulfill tasks i or j without using arc (i, j). In addition, we forbid the PA to use any arc starting at node i and ending at any node different than j. We also forbid the PA to use any

arc ending at node j and starting at any node different than i. The second level branches on walking flow variables. Similarly, we branch on the implicit variable h_{ij} that takes the value of 1 if any team walks through arc $(i, j) \in \mathcal{A}$, and 0 otherwise. Decisions are enforced in the same manner as for the first level.

The third level branches on the assignment of tasks to workers. More specifically, we branch on the implicit variables y_i^k which take 1 if worker $k \in \mathcal{W}$ fulfills task $i \in \mathcal{C}$. To enforce $y_i^k = 0$ we remove all the paths that fulfill task i and include worker k in the team. Also, we forbid the PA to fulfill task i if the worker is currently in the team. To enforce $y_i^k = 1$ we remove all the paths that fulfill task i without including worker k in the team. In addition, we remove the possibility of outsourcing task i and we forbid the PA to fulfill task i using teams that do not include worker k. The fourth level branches on the selection of workers. More specifically, we branch on the implicit variables v_k which equals 1 if worker $k \in \mathcal{W}$ is included in any path of the current solution, and 0 otherwise. To guarantee that $v_k = 0$, we remove all the paths that include worker k in the team and remove the corresponding node from the modified network. To enforce $v_k = 1$, we add a constraint to the master problem. The fifth level branches on the ϑ_i variables to either forbid or enforce the outsourcing of task $i \in \mathcal{C}$. To enforce this branching rule, we add a constraint to the master problem. The algorithm explores the enumeration branch-and-bound tree using a best-bound search strategy.

5. Computational experiments

The proposed BPC algorithm was implemented in Java using the jORLib[1] library and compiled using Java 1.8.0 331. We rely on CPLEX 20.1 to solve the master problem. All the experiments were conducted on the Beluga cluster of the Digital Research Alliance of Canada using eight threads and 20GB of RAM in a Linux environment. The time limit for all the experiments is 2 hours. After fine tuning, we set the bound step size in the PA to 10. The bounding time limits are set to 0.2ϕ and ϕ. The maximum number of paths Υ is set to 10 and the time limit Λ to 5 seconds. All the instances and solutions are available at `https://chairelogistique.hec.ca/en/scientific-data/`.

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

5.1. Set of instances

We created a set of instances based on the testbed designed by Kovacs et al. (2012) for the STRSP. These instances were derived from the well-known VRPTW instances proposed by Solomon (1987). There are three classes of instances (`D`), denoted as random (R), clustered (C), and semi-clustered (RC). For each class of instances, two types of planning horizons (`T`) were considered, namely, short (1) and long (2). In addition, the percentage of tasks that have a time window constraint (`m`) can take two values, where 01 is used to indicate 100% and 03 to specify 50%. To fulfill tasks, two sets of workers (`E`) are considered: complete (C) and reduced (R). These sets differ in the number of workers available. In the complete set, up to 130 workers are available. In the reduced set, up to 25. The number of skills domains (`S`) is selected in the set $\{5, 6, 7\}$ and each skill may have different levels of proficiency (`L`) selected from the set $\{4, 6\}$. Finally, the number of tasks (`n`) is selected in the set $\{25, 50, 75\}$. A total of 162 instances are considered. An instance is referred to as "`DTm_E_S`×`L_n`".

For each instance, the skills requirements ν_{iql}, the time window $[a_i, b_i]$, the outsourcing cost f_i, and service time associated with each task $i \in \mathcal{C}$ are known. Similarly, information regarding the qualifications ξ_{kql} of each worker $k \in \mathcal{W}$ is given. The driving distance (in kilometers) μ_{ij} between nodes (tasks and depot) is computed using the Euclidean distance. The maximum duration of each route corresponds to the latest arrival time to the depot $b_{\underline{0}}$. To extend this set of instances to the WSRP-PL, we compute driving and walking times considering a driving speed of 60 km/h and a walking speed of 4 km/h. We assume that driving and walking distances are equal. The maximum number of workers per team can be either two or three and we fixed $|\mathcal{T}| = m$. In addition, we consider a maximum daily walking distance ζ for each worker of 5 km. The maximum walking distance between two nodes θ is set to 2.5 km. Finally, the variable cost c^v is set to 1.

5.2. Experiment 1: assessing the BPC performance

To assess the effectiveness and efficiency of our BPC algorithm, we present a comparison with the solution of the arc-based formulation described in Section 2, which is the only other exact method available to solve the WSRP-PL. This formulation was solved using CPLEX 20.1.

Tables 1 and 2 present the detailed results of the comparison between the BPC algorithm and the arc-based formulation (AF) setting the maximum number of workers in a team to 2 and 3, respectively. Column 1 denotes the

set of workers. Column 2 provides information regarding the instance class and the percentage of tasks having a time window. Column 3 gives the number of tasks. The remaining columns give the number of optimal solutions found, the average optimality gap, and the computational time in seconds used by each algorithm. To compute the optimality gap we use the best lower bound known for each instance (i.e., the highest lower bound between the bounds found by AF and BPC). Whenever AF ran out of memory, we used the last integer solution found to compute the optimality gap.

Table 1: Comparison between AF and BPC on the WSRP-PL instances with $\omega = 2$.

E	Tm	n	AF			BPC		
			#Opt.	Avg. Δ	CPU (s)	#Opt.	Avg. Δ	CPU (s)
C	101	25	8/9	0.67%	1122.25	9/9	0.00%	293.01
	103	25	4/9	16.44%	7200.00	9/9	0.00%	254.34
	201	25	7/9	1.89%	2469.41	9/9	0.00%	204.36
	101	50	7/9	8.56%	3792.39	8/9	0.53%	1222.96
	103	50	0/9	38.49%	7200.00	7/9	12.69%	2115.40
	201	50	5/9	16.24%	5911.66	9/9	0.00%	1188.62
	101	75	4/9	29.70%	6126.59	9/9	0.00%	1784.40
	103	75	0/9	53.18%	7200.00	6/9	13.64%	3618.37
	201	75	1/9	32.29%	6411.02	5/9	15.28%	3788.10
R	101	25	9/9	0.00%	199.75	9/9	0.00%	76.25
	103	25	6/9	2.82%	6859.72	9/9	0.00%	834.51
	201	25	9/9	0.00%	244.87	9/9	0.00%	172.01
	101	50	8/9	0.37%	3134.59	9/9	0.00%	498.68
	103	50	0/9	7.86%	7200.00	7/9	2.27%	2399.45
	201	50	7/9	3.14%	2161.92	7/9	5.04%	1628.97
	101	75	5/9	9.95%	4172.81	7/9	0.23%	1825.47
	103	75	0/9	31.45%	7200.00	5/9	12.03%	3999.66
	201	75	4/9	10.90%	5633.11	7/9	22.22%	2334.37
Total/Avg.			84/162	14.66%	4680.01	140/162	4.66%	1568.83

Table 1 shows that BPC can solve 140 out of 162 instances to optimality, while AF can only solve 84. Remarkably, BPC can solve all the instances with 25 tasks to optimality. As expected, instances in which the percentage of tasks having a time window is lower (i.e., Tm = 103) are harder to solve. This is especially the case for AF, as it can only solve 10 out of 54 instances with this setting. A similar argument can be used for the subset of instances

with wider time windows (i.e., Tm = 201). As routes are longer, BPC, a column generation-based method, decreases its performance. With regard to the number of workers available, it seems that AF better handles instances with a lower number of workers (i.e., E = R), as it can solve 12 more instances (48 vs 36) compared with the subset of instances with the complete set of workers. One plausible explanation is that the reduction in the number of variables has a strong positive impact on AF's performance. With respect to the computational times, on average BPC takes 1568.83 seconds to solve the WSRP-PL, while AF takes 4680.01 seconds. BPC is particularly fast on the subset of instances with 25 customers.

Table 2: Comparison between AF and BPC on the WSRP-PL instances with $\omega = 3$.

E	Tm	n	AF			BPC		
			#Opt.	Avg. Δ	CPU (s)	#Opt.	Avg. Δ	CPU (s)
C	101	25	8/9	1.10%	1471.38	8/9	2.20%	2875.83
	103	25	2/9	19.90%	7200.00	7/9	7.77%	3902.10
	201	25	7/9	4.30%	2681.78	7/9	2.94%	3085.69
	101	50	4/9	14.02%	5113.89	4/9	2.84%	5498.06
	103	50	0/9	63.03%	7200.00	3/9	22.51%	6131.62
	201	50	2/9	14.88%	5839.18	4/9	7.75%	4968.35
	101	75	2/9	29.36%	6602.18	4/9	12.37%	6323.48
	103	75	0/9	80.21%	7200.00	2/9	45.81%	6667.82
	201	75	1/9	36.55%	6419.64	2/9	15.63%	6837.68
R	101	25	9/9	0.00%	396.73	9/9	0.00%	238.93
	103	25	5/9	3.16%	7200.00	9/9	0.00%	495.01
	201	25	9/9	0.00%	511.32	9/9	0.00%	518.86
	101	50	6/9	1.20%	4192.70	8/9	1.81%	1835.20
	103	50	0/9	14.40%	7200.00	5/9	7.80%	3668.69
	201	50	4/9	1.61%	4191.36	5/9	3.89%	3310.46
	101	75	2/9	14.51%	6424.36	3/9	4.22%	5163.17
	103	75	0/9	54.11%	7200.00	1/9	26.65%	6814.85
	201	75	3/9	14.63%	5429.80	2/9	13.78%	6033.96
Total/Avg.			64/162	20.39%	5137.46	92/162	9.89%	4131.65

Table 2 shows that BPC can solve 92 out of 162 instances to optimality, while AF only solves 64. The average optimality gap of the solutions retrieved by BPC is 9.89% while the average optimality gap of the solutions found

by AF is 20.39%. With respect to the computational times, on average BPC takes 4131.65 seconds to solve the WSRP-PL, while AF takes 5137.46 seconds. Note, that increasing the maximum number of workers per team increases the difficulty of the problem. This is expected because the possible number of teams that can be formed is significantly larger.

5.3. Experiment 2: Analyzing the computational impact of park-and-looping

We set up an experiment to shed some light into the computational impact of allowing park-and-loop routes. To accomplish this goal we ran our BPC on the whole set of instances setting $\zeta = 0$ (i.e., forbidding the walking subtours). Next, we compared the results to those obtained when park-and-looping is allowed (see Experiment 1). Table 3 summarizes the results delivered by BPC running with (BPC) and without (BPC-D) park-and-looping. Each row corresponds to a combination of a set of workers, the maximum number of workers per team, and the number of tasks. Columns 4 and 7 report the number of optimal solutions found in each setting. Columns 5 and 8 show the average optimality gap. Columns 6 and 9 contain the average CPU time in seconds.

Table 3: Comparison of BPC performance on the WSRP-PL with and without allowing park-and-loop routes.

E	Tm	n	BPC-D			BPC		
			#Opt.	Avg. Δ	CPU (s)	#Opt.	Avg. Δ	CPU (s)
C	101	25	18/18	0.00%	773.75	17/18	1.10%	1584.42
	103	25	17/18	0.14%	733.82	16/18	3.89%	2078.22
	201	25	18/18	0.00%	835.73	16/18	1.47%	1645.02
	101	50	16/18	1.19%	2036.43	12/18	1.68%	3360.51
	103	50	13/18	5.9%	3038.96	10/18	17.60%	4123.51
	201	50	17/18	0.42%	1730.15	13/18	3.88%	3078.49
	101	75	17/18	0.88%	2359.70	13/18	6.19%	4053.94
	103	75	9/18	12.83%	4425.10	8/18	29.73%	5143.10
	201	75	10/18	4.56%	4791.78	7/18	15.46%	5312.89
R	101	25	18/18	0.00%	91.39	18/18	0.00%	157.59
	103	25	18/18	0.00%	270.48	18/18	0.00%	664.76
	201	25	18/18	0.00%	60.89	18/18	0.00%	345.4
	101	50	18/18	0.00%	850.00	17/18	0.90%	1166.94
	103	50	15/18	1.15%	1908.46	12/18	5.04%	3034.07
	201	50	14/18	3.27%	2108.47	12/18	4.46%	2469.72
	101	75	11/18	2.27%	3822.91	10/18	2.23%	3494.32
	103	75	8/18	19.27%	4624.48	6/18	19.34%	5407.26
	201	75	12/18	8.35%	3335.90	9/18	18.00%	4184.16
Total/Avg.			267/324	3.35%	2099.91	232/324	7.28%	2850.24

The results indicate that allowing park-and-looping significantly increases the difficulty of the problem. Out of the 324 instances, BPC-D proves optimality on 267, while BPC achieves it in 232. Moreover, BPC-D is able to establish optimality in all 25-task instances and in 95 out of 108 of the 50-task instances. A closer examination of the optimality gaps and CPU times confirms the conclusion. While BPC-D reports an average gap of 3.35%, BPC reports a nearly twofold average gap of 7.28%. The former also runs (on average) nearly 30% faster (2099.91 vs. 2850.24 seconds).

5.4. Experiment 3: Measuring the impact of the subtour transportation mode

In this experiment, we aim to measure the impact of using a faster and longer-range transportation mode (e.g., an electric bike or scooter) to perform the subtours. More specifically, we compare the solutions we obtained in Experiment 1 with those obtained for each instance while increasing the

walking speed (10 km/h) and the maximum walking distance (10 km) to mimic the behavior of an electric scooter.

Table 4 compares the solutions found by BPC for each instance when considering both configurations. We denote the configuration that mimics the e-scooter behavior as BPC-S. The first three columns provide information regarding the set of workers, the instance series, and the number of customers. Columns 4 and 9 report the number of optimal solutions found by BPC under each configuration. Columns 5 and 10 present the average optimality gap. The optimality gap is computed using the lower bound found by each algorithm. Columns 6 and 11 report the average computational time in seconds. Columns 7 and 12 present the average number of subtours in each solution. Columns 8 and 13 report the average maximum walking distance. Finally, Column 14 presents the average gap in the objective function when both BPC and BPC-S solved the instance to optimality. More specifically, for each instance in the set, the gap was computed as

$$\frac{f(BPC\text{-}S) - f(BPC)}{f(BPC)}, \tag{45}$$

where $f(\cdot)$ is the objective function found by BPC under each configuration.

Table 4: Comparison of BPC performance while varying the walking speed and the maximum walking distance.

E	Tm	n	BPC					BPC-S					Δ OF
			#Opt.	Avg. Δ	CPU (s)	#S.	WD	#Opt.	Avg. Δ	CPU (s)	#S.	WD	
C	101	25	17/18	1.10%	1584.42	0.72	2.39	13/18	3.75%	2776.43	1.72	12.14	-0.49%
	103	25	16/18	3.89%	2078.22	1.06	2.25	13/18	16.53%	3013.13	2.11	11.78	-0.72%
	201	25	16/18	1.47%	1645.02	0.61	1.33	14/18	7.32%	2948.16	3.39	17.20	-2.66%
	101	50	12/18	1.68%	3360.51	0.56	1.16	10/18	10.43%	4312.47	3.22	13.22	-0.54%
	103	50	10/18	17.60%	4123.51	0.78	1.14	8/18	30.23%	4985.41	2.56	12.42	-0.69%
	201	50	13/18	3.88%	3078.49	0.67	1.56	7/18	20.30%	4902.07	3.83	17.69	-1.83%
	101	75	13/18	6.19%	4053.94	0.67	1.19	7/18	18.22%	5009.48	4.17	15.90	-0.85%
	103	75	8/18	29.73%	5143.10	0.41	0.50	3/18	50.73%	6204.16	2.33	10.94	-0.19%
	201	75	7/18	15.46%	5312.89	0.44	1.11	5/18	35.36%	6428.05	2.56	12.36	-0.59%
R	101	25	18/18	0.00%	157.59	0.78	2.61	17/18	0.03%	867.18	2.50	13.48	-1.12%
	103	25	18/18	0.00%	664.76	1.11	2.27	17/18	0.27%	1101.10	2.61	12.57	-2.75%
	201	25	18/18	0.00%	345.43	0.83	2.00	18/18	0.00%	461.03	3.83	19.35	-3.35%
	101	50	17/18	0.90%	1166.94	0.72	1.39	14/18	1.35%	2229.40	3.67	14.94	-0.58%
	103	50	12/18	5.04%	3034.07	1.11	1.80	12/18	12.80%	3786.36	3.89	16.69	-0.90%
	201	50	12/18	4.46%	2469.72	0.72	1.78	11/18	15.92%	3484.28	4.39	17.86	-0.61%
	101	75	10/18	2.23%	3494.32	0.78	1.39	10/18	4.85%	4084.56	4.22	12.58	-0.42%
	103	75	6/18	19.34%	5407.26	0.67	0.72	5/18	23.72%	5514.21	2.28	9.91	-0.34%
	201	75	9/18	18.00%	4184.16	0.69	1.25	5/18	19.26%	5611.14	3.17	15.73	-0.44%
Total/Avg.			232/324	7.28%	2850.24	0.74	1.55	189/324	15.06%	3762.15	3.14	14.26	-1.06%

As the results show, increasing the speed and the range of the mode employed for the subtours has a positive impact on the objective function. On average, the objective function decreases by 1.06%, but the savings are probably greater. Note that BPC-S only solves to optimality 189 out of 324 instances while BPC solves 232 out of 324. As a result, the average optimality gap reported by BPC-S is 15.06%, while BPC solutions have an average optimality gap of 7.28%. These figures also indicate that solving an instance of the WSRP-PL under the second configuration is significantly harder. A possible explanation for this behavior is given by the average number of subtours performed in each route under both settings. Indeed, while BPC finds solutions with an average of 0.74 subtours, BPC-S builds solutions with 3.14 subtours on average.

5.5. Experiment 4: solving the no-team STRSP

In Experiment 1, we demonstrated that our BPC can optimally solve WSRP-PL instances with up to 75 nodes. The most closely related problem, featuring publicly available instances of comparable size, is the no-team version of the STRSP. Note that our WSRP-PL reduces to that problem when $\omega = 1$ and $\zeta = 0$. We, therefore, ran our AF and BPC on the set of large 100-task instances proposed by Kovacs et al. (2012), available at `http://prolog.univie.ac.at/research/STRSP/` and compared our results to the state of the art.

Table 5 and 6 compare the performance of BPC and AF on the no-team STRSP instances when the set of workers is complete and reduced, respectively. Columns 1 and 2 denote the class of the instance and the instance series. Column 3 presents the average number of workers available. Columns 4 and 7 report the number of optimal solutions found by each algorithm. Columns 5 and 8 show the average optimality gap. Columns 6 and 9 contain the average CPU time employed by each algorithm in seconds.

Table 5: Comparison between AF and BPC on the Kovacs et al. (2012) 100-task complete instances of the no-team STRSP.

D	Tm	$\|\mathcal{W}\|$	AF			BPC		
			#Opt.	Avg. Δ	CPU (s)	#Opt.	Avg. Δ	CPU (s)
C	101	16.7	0/3	29.10%	7200.0	3/3	0.00%	929.9
R	101	26.3	0/3	6.87%	7200.0	3/3	0.00%	717.9
RC	101	23	0/3	40.68%	7200.0	3/3	0.00%	1027.5
C	103	16.7	0/3	33.16%	7200.0	0/3	4.74%	7200.0
R	103	26.3	0/3	96.59%	7200.0	1/3	0.36%	4929.0
RC	103	23	0/3	32.85%	7200.0	0/3	4.97%	7200.0
C	201	7.67	0/3	8.06%	7200.0	2/3	5.29%	2425.5
R	201	8	0/3	0.40%	7200.0	1/3	17.79%	4855.9
RC	201	8.67	0/3	0.63%	7200.0	0/3	6.37%	7200.0
Total/Avg.			0/27	27.6%	7200.0	13/27	4.39%	4054.0

Table 6: Comparison between AF and BPC on the Kovacs et al. (2012) 100-task reduced instances of the no-team STRSP.

D	Tm	$\|\mathcal{W}\|$	AF			BPC		
			#Opt.	Avg. Δ	CPU (s)	#Opt.	Avg. Δ	CPU (s)
C	101	4.67	0/3	16.47%	7200.0	3/3	0.00%	102.7
R	101	4.67	0/3	33.43%	7200.0	3/3	0.00%	398.1
RC	101	4.67	0/3	53.65%	7200.0	3/3	0.00%	3514.4
C	103	4.67	0/3	0.95%	7200.0	1/3	14.84%	6750.8
R	103	4.67	0/3	94.01%	7200.0	1/3	3.62%	4835.2
RC	103	4.67	0/3	62.99%	7200.0	0/3	16.52%	7200.0
C	201	4.67	2/3	0.05%	2716.8	0/3	18.38%	7200.0
R	201	4.67	0/3	0.41%	7200.0	0/3	40.75%	7200.0
RC	201	4.67	0/3	0.48%	7200.0	0/3	29.66%	7200.0
Total/Avg.			2/27	29.16%	6701.9	11/27	13.75%	4933.5

Tables 5 and 6 clearly indicate that in this problem our BPC also outperforms AF. In the subset of instances in which the complete set of workers is available, BPC solves 13 out of 27 instances to optimality, while AF solves none. The solution found by BPC improves the best known solution reported by Gu et al. (2022) in 3 of these 13 instances. Moreover, the average optimality gap of the solutions found by BPC is 4.39% while AF reports solutions with an average optimality gap of 27.6%. With respect to the computational

times, AF always reached the time limit. In contrast, BPC uses on average 4054 seconds. A similar situation is observed in the subset of instances with a reduced set of workers. AF solves 2 out of 27 instances to optimality, while BPC solves 11. The solution found by BPC is the new best known solution for 9 of these instances. On average the optimality gap of the solutions provided by AF is 29.16% while BPC finds solutions with an average optimality gap of 13.75%. The objective function of the new best known solutions found by BPC is available in Appendix A.

6. Concluding remarks

In this paper, we introduced the workforce scheduling and routing problem with park-and-loop. Routing decisions can be particularly challenging as routes can include one or more subtours that are covered on foot. To solve this problem we presented a branch-price-and-cut algorithm that is capable of solving instances with up to 75 tasks and 130 workers in less than two hours. The algorithm also provided optimal solutions for the closely related technician routing and scheduling problem.

Extensive computational experiments carried out on instances derived from a popular benchmark in the literature suggest that the proposed algorithm can outperform a standard arc-based formulation both in terms of solution quality and running times. Indeed, our algorithm solved 232 out of 324 instances to optimality while the arc-based formulation only solved 148. This performance gap comes as a result of using state-of-the-art techniques and exploiting problem-specific features to enhance the pricing problem algorithm.

To encourage further research on the WSRP-PL we designed an online tool where all the instances and solutions are available. Further research will focus on a more challenging variant of the problem in which teams can partially split (i.e., workers can fulfill a task on their own and then rejoin the team). In some cases, splitting a team may be beneficial to the company's efficiency. However, it can easily disrupt the structure of the routes, that do not longer follow a regular park-and-loop structure.

Acknowledgements

This research was partially funded by the Canada First Research Excellence Fund through IVADO and the Discovery grants. This research was also

co-funded by HEC Montréal through the Chair in Logistics and Transportation and the Research Professorship on Clean Transportation Analytics.

References

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

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

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

Cabrera, N., Cordeau, J.-F., & Mendoza, J. E. (2023). Solving the park-and-loop routing problem by branch-price-and-cut. *Transportation Research Part C: Emerging Technologies*, *157*, 104369.

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

Castillo-Salazar, J. A., Landa-Silva, D., & Qu, R. (2016). Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, *239*, 39–67.

Chen, X., Thomas, B. W., & Hewitt, M. (2016). The technician routing problem with experience-based service times. *Omega*, *61*, 49–61.

Coindreau, M.-A., 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 multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, *12*, 129–146.

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

Dellaert, N., Dashty Saridarq, F., Van Woensel, T., & Crainic, T. G. (2019). Branch-and-price–based algorithms for the two-echelon vehicle routing problem with time windows. *Transportation Science*, *53*, 463–479.

Goel, A., & Meisel, F. (2013). Workforce routing and scheduling for electricity network maintenance with downtime minimization. *European Journal of Operational Research*, *231*, 210–228.

Gu, H., Zhang, Y., & Zinder, Y. (2022). An efficient optimisation procedure for the workforce scheduling and routing problem: Lagrangian relaxation and iterated local search. *Computers & Operations Research*, *144*, 105829.

Guastaroba, G., Côté, J.-F., & Coelho, L. C. (2021). The multi-period workforce scheduling and routing problem. *Omega*, *102*, 102302.

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

Kovacs, A. A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, *15*, 579–600.

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*, *308*, 1233–1248.

Lozano, L., Duque, D., & Medaglia, A. L. (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, *50*, 348–357.

Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*, 378–384.

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 & Operations Research*, *114*, 104833.

Mendoza, J. E., & Villegas, J. G. (2013). A multi-space sampling heuristic for the vehicle routing problem with stochastic demands. *Optimization Letters*, *7*, 1503–1516.

Paraskevopoulos, D. C., Laporte, G., Repoussis, P. P., & Tarantilis, C. D. (2017). Resource constrained routing and scheduling: Review and research prospects. *European Journal of Operational Research*, *263*, 737–754.

Pereira, D. L., Alves, J. C., & de Oliveira Moreira, M. C. (2020). A multiperiod workforce scheduling and routing problem with dependent tasks. *Computers & Operations Research*, *118*, 104930.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, *35*, 254–265.

Tricoire, F., Bostel, N., Dejax, P., & Guez, P. (2013). Exact and hybrid methods for the multiperiod field service routing problem. *Central European Journal of Operations Research*, *21*, 359–377.

Xie, F., Potts, C. N., & Bektaş, T. (2017). Iterated local search for workforce scheduling and routing problems. *Journal of Heuristics*, *23*, 471–500.

Zamorano, E., Becker, A., & Stolletz, R. (2018). Task assignment with start time-dependent processing times for personnel at check-in counters. *Journal of Scheduling*, *21*, 93–109.

Zhou, Y., Huang, M., Wu, H., Chen, G., & Wang, Z. (2020). Iterated local search with hybrid neighborhood search for workforce scheduling and routing problem. In *2020 12th International Conference on Advanced Computational Intelligence (ICACI)* (pp. 478–485). IEEE.

Appendix A. New best known solutions for the no-team STRSP

Table A.7 compares the solution found by BPC and the best solution reported in the literature. Column 1 shows the instance identifier. Column 2 shows the objective function of the best known solution reported by Gu et al. (2022). Column 3 reports the objective function of the solution found by BPC.

Table A.7: New best known solutions for the no-team STRSP 100-task instances.

Instance	L-ILS	BPC
RC101_C_5x4_100	1658.37	1654.80
RC101_C_6x6_100	1654.98	1644.32
R103_C_7x4_100	1335.65	1325.31
C101_R_5x4_100	5587.52	5572.99
RC101_R_5x4_100	4829.37	4764.44
C103_R_6x6_100	4804.61	4799.01
R101_R_6x6_100	5945.14	5944.91
RC101_R_6x6_100	4903.70	4835.20
C101_R_7x4_100	5241.93	5208.30
C103_R_7x4_100	1980.72	1940.21
R103_R_7x4_100	2104.89	2104.17
RC103_R_7x4_100	2585.95	2568.25