Publié par :
Published by:
Publicación de la:

Faculté des sciences de l'administration
2325, rue de la Terrasse
Pavillon Palasis-Prince, Université Laval
Québec (Québec) Canada G1V 0A6
Tél. Ph. Tel. : (418) 656-3644
Télec. Fax : (418) 656-7047

Disponible sur Internet :
Available on Internet
Disponible por Internet :

http://www4.fsa.ulaval.ca/la-recherche/publications/documents-de-travail/

DOCUMENT DE TRAVAIL 2019-017

The Multi-Period Workforce Scheduling and Routing Problem

GianFranco GUASTAROBA
Jean-François CÔTÉ
Leandro C. COELHO

Document de travail également publié par le Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, sous le numéro CIRRELT-2019-54

Novembre 2019

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2019
Bibliothèque et Archives Canada, 2019

ISBN 978-2-89524-496-7 (PDF)

The Multi-Period Workforce Scheduling and Routing Problem

Gianfranco Guastaroba[1], **Jean-François Côté**[2, *], **Leandro C. Coelho**[2]

[1] Department of Economics and Management, University of Brescia, Brescia, Italy
[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Operations and Decision Systems, 2325, rue de la Terrasse, Université Laval, Québec, Canada, G1V 0A6

**Corresponding author: jean-francois.cote@cirrelt.ca*

ABSTRACT

Motivated by the problem faced by a company that provides installation and maintaining services of electrical infrastructures, we study the Multi-Period Workforce Scheduling and Routing Problem (MPWSRP). A set of jobs requiring a service is given. A given set of workers, each proficient in a number of skills, is available to serve the jobs. Each job requires a team of workers providing the required skills, and is associated with a given priority level. A service time, which might span over multiple time periods, is associated with each job, and depends on the number of workers that have been assigned. The MPWSRP calls for determining an optimal dispatch plan to serve the given set of jobs by routing a number of teams over a finite planning horizon. The MPWSRP is a highly complex combinatorial optimization problem where routing, scheduling, and assignment decisions have to be taken simultaneously. We cast the MPWSRP as a Mixed-Integer Linear Program (MILP). As an off-the-shelf solver can solve only small-size instances of the proposed MILP in reasonable computing times, we devise an Adaptive Large Neighborhood Search (ALNS) heuristic to solve large-size instances. Extensive computational experiments are conducted on a large set of instances. The results show that ALNS is competitive with the solver on the small-size instances, producing solutions of similar average quality in considerably shorter computing times. Further computational experiments are conducted applying our ALNS on a set of large-size instances, as well as real data sets.

Keywords: Workforce routing and scheduling, multi-period planning, mixed-integer linear programming, adaptive large neighborhood search, matheuristic.

Acknowledgments: We wish to thank our contact persons at the company partner for providing us with the data necessary to conduct the case study. This work was partly funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 2015-04893 and 2019-00094. This support is gratefully acknowledged.

1 Introduction

In the services industry, it is crucial to effectively manage a crew of skilled workers to perform installation, construction, and maintenance activities. Workers have different skills and can be combined together to compose heterogeneous teams. These teams must be dispatched to serve some jobs. Jobs can be simple and quick, requiring few workers and lasting for only some minutes, or very complex, requiring many workers and lasting for several days. The scheduling decisions of when to serve each job over a discrete planning horizon is a difficult task, which must be coupled with the composition of teams (by assigning workers to each of them) and the routing of each team on each day from a depot to serve the set of jobs they have been assigned to. This is a very challenging integrated vehicle routing problem, where routing, scheduling, and assignment decisions have to be taken simultaneously. In the academic community, integrated vehicle routing problems is the expression frequently used to denote the class of problems where a Vehicle Routing Problem (VRP) arises in combination with other optimization problems within the broader context of logistics and transportation (e.g., cf. Bektaş et al., 2015). This class of problems is recently attracting an ever growing attention among academics and practitioners. Côté et al. (2017) state that this increasing body of literature is motivated by, on the one side, the desire of bridging the gap between academic problems and real-world applications, and, on the other side, the recent advances in optimization methods and computer capabilities that are making it possible to jointly solve strongly interdependent problems that have been, until recently, addressed independently.

Inspired by a real-world problem faced by a service company operating in the province of Québec, Canada, we study the Multi-Period Workforce Scheduling and Routing Problem (MPWSRP). This company installs, maintains, and repairs electrical infrastructures. Each job accepted by the company requires a given set of skills. Jobs can be simple and quick (e.g., installing a new power outlet), or very complex and long (e.g., wiring a new building). Jobs also have a priority, as some are more urgent than others. The company has hired a group of skilled workers to serve the jobs, and has a limited fleet of vehicles. The service time of each job depends on the number of workers assigned to it (e.g., three workers can finish a given job in a shorter time than two workers). At the beginning of the planning horizon, the company has to decide when each job will be served. Simultaneously, the company, has to decide how to group workers into teams, how to assign each team to a set of jobs, and to determine a routing plan for each team on each day.

The use of skilled workers is common in the literature. In what follows, we survey the most relevant contributions to our application, and refer the reader interested in workforce planning problems incorporating skills to the survey by Bruecker et al. (2015) and, and if routing aspects are to be considered, to the overview provided by Castillo-Salazar et al. (2016). Li et al. (2005) consider different skills for a set of workers and link them to a job by using constraints to assign a team to a job. Each job has a fixed service time, and the routing aspect considers time windows. The problem is defined on a single period, just like that of Xu and Chiu (2001), who considers also job priorities, imposing that some jobs must be performed prior to others. Kim et al. (2010) handle priorities as precedence constraints between jobs, proposing a mathematical formulation and solving the problem heuristically. Dohn et al. (2009) do not consider job priorities, but allow jobs to be left unassigned, and solve the problem via a branch-and-price algorithm. Kovacs et al. (2012) provide some flexibility by introducing the possibility of not serving some jobs, but rather outsourcing them. They propose a mathematical formulation and solve the problem using an Adaptive Large Neighborhood Search (ALNS) heuristic. Finally, Pillac et al. (2013) consider a problem in which the usage of tools and spare parts are also optimized, other than routing a set of skilled technicians, solving it via a hybrid of mathematical programming and ALNS.

To the best of our knowledge, the only multi-period problem similar to ours is that of Tang et al. (2007), but their version does not consider a skilled workforce neither job priorities. They solve the problem via a tabu search algorithm. Cortés et al. (2014) also do not consider the workforce in their problem, but jobs have priorities. Their branch-and-price algorithm allows jobs to be left unserved, and those serviced are considered with soft time windows. Goel and Meisel (2013) considers explicitly crews but not their skills, assuming the workforce to be homogeneous.

Other types of problems involving scheduling of heterogeneously skilled workers include call center scheduling, such as in Legros et al. (2015) who consider different workloads and requirements for the operators, and different service requirements on the demand side (unbalanced demand, for example). In the home healthcare service industry, one has to route workers to the patients houses, and each patient may require the visit of a different type of worker, configuring skills. Here, the service is typically of short duration and repetitive, allowing for some cyclic operations. Examples in this area include the works of Restrepo et al. (2019); Castaño and Velasco (2019). In the retail industry, employee scheduling is a critical task, and in some cases skills are also present (e.g., cf. Bürgy et al., 2019). In postal services, the problem also arises with a routing component, such as in the problem tackled by Brunner and Bard (2013).

Several features of the different variants mentioned above are considered simultaneously in our problem, which is motivated by our observation in the industry, as detailed in the following.

Contributions. We introduce a new problem in the class of routing and scheduling problems of skilled workforce crews. The main original characteristics of our problem are the multi-period setting, the presence of jobs whose service times span across multiple time periods, and service times that are a function of the team composition. The setting of our problem is very flexible, as are operations in the field. Teams can be changed from one day to another, and different workers might perform the same job across multiple periods, as long as they have the required skills to perform it. Once started, a job must be continually carried out even when spanning multiple days, meaning that once the job is started, a team can only leave it when their work shift ends, and the same job will be executed as the first activity on the next day by the same or another team. We propose an innovative Mixed-Integer Linear Programming (MILP) formulation for the MPWSRP in which teams are predetermined by enumeration, and their assignment to time periods and jobs is done optimally via mathematical programming. To tackle large-size instances of the MPWSRP, we propose a matheuristic based on ALNS in which some of its operators are based on the exact solution of some easy-to-solve mathematical models. Extensive computational experiments are conducted on a large set of instances. The results show that an off-the-shelf solver can solve in reasonable computing times only small-size instances of the proposed MILP. Our ALNS is competitive with the solver on the small-size instances, producing solutions of similar average quality in considerably shorter computing times. We run some further computational experiments by solving large-size instances with ALNS. Finally, we provide a case study based on the data provided by our industrial partner.

Structure of the paper. The remainder of the paper is organized as follows. In Section 2 we provide a formal description of the MPWSRP along with a MILP formulation. Section 3 describes the implementation of our solution algorithm and its internal operators specifically designed to tackle the problem at hand. The results of the extensive computational experiments are described in Section 4. Finally, some concluding remarks are outlined in Section 5.

2 Problem description and mathematical formulation

We consider a multi-period setting defined over a finite planning horizon. The planning horizon is discretized into a number T of time periods, e.g., workdays. Let $\mathcal{T} = \{1, \ldots, T\}$ denote the set of

time periods. To simplify the description, henceforth we refer to a time period simply as a day. The MPWSRP can be defined as follows. We are given a set of jobs $J = \{1, \ldots, n\}$ and a set of workers $W = \{1, \ldots, |W|\}$. Each job is associated with a location and all workers begin their day at a depot. The MPWSRP can be represented on a graph $G = (V, A)$, where $V = \{0, n+1\} \cup J$ represents the set of vertices with cardinality $|V| = |J| + 2$, and vertices 0 and $n+1$ correspond to the start and end depot, possibly representing the same physical location. Set $A = \{(i, j) : i, j \in V, i \neq j\}$ contains the arcs connecting pairs of vertices. A nonnegative traveling cost (time) c_{ij} (t_{ij}) is associated with each arc $(i, j) \in A$, and we assume that both c_{ij} and t_{ij} satisfy the triangle inequality. At the end of each day, all workers have to return to the end depot. Hence, each route is constrained to start at vertex 0 and to end at vertex $n+1$.

Each worker $w \in W$ is specialized in a number of skill domains. We consider that S skill domains exist (i.e., are required to satisfy all jobs) and represent each worker's proficiency in these domains by an $S-$dimensional binary vector in which the s^{th} entry $p_{ws} \in \{0, 1\}$ takes value 1 if the worker w is skilled in the s^{th} domain, and 0 otherwise. All the other characteristics, such as service times, are assumed to be homogeneous among workers. It is worth noting that some authors (e.g., cf. Cordeau et al., 2010; Kovacs et al., 2012) consider that within each skill domain, workers can have different proficiency levels. We do not explicitly consider different proficiency levels since we have observed that, in practice, workers have usually a card indicating whether they are able to perform a given activity, without detailing the proficiency level. Nevertheless, the presence of different proficiency levels can be captured within our approach by simply replicating, for each skill, a binary parameter for as many proficiency levels as needed.

At the beginning of each day t, with $t = 1, \ldots, T$, the set W of workers is partitioned into subsets called teams. Let K denote the set of all possible teams of workers that can be created. Each team $k \in K$ then departs from depot 0 at the beginning of the day (i.e., time 0), serves some jobs (the first and the last ones possibly partially), and then returns to the depot $n+1$ before the end of the daily work shift. Let $\overline{T}$ be the length of the daily work shift. Note that, in the following, the terms team, vehicle and route are used interchangeably as a team is assigned to a vehicle, which is assigned to a route; and, conversely, a route is only assigned to a vehicle, which is only assigned to a team. Note also that the team composition, once decided, cannot be changed within a day, but can be different from one day and another.

Each job $j \in J$ requires a set of skills to be completed. Similarly to workers' proficiencies, we represent the requirements for each job by an $S-$dimensional binary vector in which the s^{th} entry $r_{js} \in \{0, 1\}$ takes value 1 if job j requires the skill in the s^{th} domain, and 0 otherwise. Each job also requires a minimum number of workers y_j^{min} to be performed. In contrast with most of the related literature, we assume that the service time required to complete a job is a function of the number of workers assigned to it. Let v_j^{max} be the service time required if we assign exactly y_j^{min} workers to carry out job j. The actual (or real) service time can be smaller than v_j^{max} if the number of workers assigned to this job is larger than the minimum. To consider this generalization, we introduce the parameter Δ_k that denotes the efficiency, in terms of reduction of the service time, associated with team k, composed of $|k|$ workers. Let v_j^{min} be the service time of job j when it is performed by the most efficient team able to perform it. Then, $b_j^{min} = \left\lceil \frac{v_j^{min}}{\overline{T} - 2t_{0j}} \right\rceil$ and $b_j^{max} = \left\lceil \frac{v_j^{max}}{\overline{T} - 2t_{0j}} \right\rceil + 1$ represent the minimum and maximum integer number of days, respectively, on which job j can be performed. Finally, each job j is also characterized by a priority level $p_j \in \mathbb{N}$ that defines its urgency. These priorities define the set of precedence constraints $P = \{(i, j) : p_i < p_j, i, j \in J\}$. In more details, these constraints impose that, if $(i, j) \in P$, job i must be at the latest started on the same day of job j, and if i and j are served by the same team, job i must be served before job j. In other words, these

priorities impose that, in any feasible solution to the MPWSRP, before a team can start serving any job with a given priority level, all jobs with a higher priority must have been started in a previous day or, at latest, on the same day. Note that this constraint requires that a job must be started, but not necessarily finished before serving any job with a lower priority. This is to avoid that, in the presence of one job whose service spans across multiple days, some teams remain idle several days waiting for that job to be finished. The restriction that if i and j are served by the same team, and $(i,j) \in P$, job i must be served before job j can be simply captured by modifying the traveling cost matrix and setting c_{ji} to a sufficiently large value for each $i,j \in J$ such that $(i,j) \in P$.

To serve the jobs, a set of D homogeneous vehicles is available. Each vehicle can accommodate at most Q workers, and all vehicles must return to the depot after at most $\overline{T}$ units of time, i.e., on each day the sum of service and traveling times for each team must be no more than $\overline{T}$.

We formulate the MPWSRP using a set of four-index vehicle flow variables to model the routing of the teams. Let $x^t_{ijk} \in \{0,1\}$ take value 1 if team k traverses arc $(i,j) \in A$ on day t. Our model also uses the following decision variables:

- $z^t_j \in \{0,1\}$ takes value 1 if the service of job j is started on day t.
- $z^t_{jk} \in \{0,1\}$ takes value 1 if job j is assigned to team k on day t.
- $g^l_j \in \{0,1\}$ takes value 1 if job j is served across l consecutive days.
- $a^t_j \geq 0$ represents the moment in time when a team arrives at job j on day t.
- $v^t_{jk} \geq 0$ represents the actual service time spent by team k serving job j on day t.

The optimization model uses also the following 0-1 parameters:

- e_{wk} takes value 1 if worker w is assigned to team k, and 0 otherwise.
- e_{jk} takes value 1 if job j can be performed by team k, and 0 otherwise.

The MPWSRP can be cast as the following MILP model:

$$\min \sum_{(i,j)\in A} c_{ij} \sum_{t\in\mathcal{T}} \sum_{k\in K} x^t_{ijk} \tag{1}$$

$$\text{s.t.} \sum_{j\in J} x^t_{0jk} \leq 1 \qquad k \in K, t \in \mathcal{T} \tag{2}$$

$$\sum_{k\in K} e_{wk} \sum_{j\in J} z^t_{jk} \leq 1 \qquad w \in W, t \in \mathcal{T} \tag{3}$$

$$\sum_{j\in J} \sum_{k\in K} x^t_{0jk} \leq D \qquad t \in \mathcal{T} \tag{4}$$

$$\sum_{i\in V, i\neq j} x^t_{ijk} - \sum_{i\in V, i\neq j} x^t_{jik} = 0 \qquad j \in J, k \in K, t \in \mathcal{T} \tag{5}$$

$$\sum_{i\in V, i\neq j} x^t_{ijk} = z^t_{jk} \qquad j \in J, k \in K, t \in \mathcal{T} \tag{6}$$

$$\sum_{t\in\mathcal{T}} z^t_j = 1 \qquad j \in J \tag{7}$$

$$\sum_{l=b_j^{min}}^{b_j^{max}} g_j^l = 1 \qquad j \in J \qquad (8)$$

$$\sum_{k\in K}\sum_{t\in\mathcal{T}} z_{jk}^t = \sum_{l=b_j^{min}}^{b_j^{max}} l g_j^l \qquad j \in J \qquad (9)$$

$$z_j^t + \sum_{l=b_j^{min}|l+t>T+1}^{b_j^{max}} g_j^l \leq 1 \qquad j \in J, t \in \mathcal{T} \qquad (10)$$

$$\sum_{k\in K} e_{jk} z_{jk}^t \leq 1 \qquad j \in J, t \in \mathcal{T} \qquad (11)$$

$$\sum_{k\in K} e_{jk} z_{jk}^{t+l} \geq z_j^t \qquad j \in J, l = 0, \ldots, b_j^{min} - 1, t \in \mathcal{T} \qquad (12)$$

$$\sum_{k\in K} e_{jk} z_{jk}^{t+l} \geq z_j^t + \sum_{\bar{l}=l+1}^{b_j^{max}} g_j^{\bar{l}} - 1 \qquad j \in J, l = b_j^{min}, \ldots, b_j^{max} - 1, t \in \mathcal{T} \qquad (13)$$

$$\sum_{k\in K} (x_{jn+1k}^{t-1} + x_{0jk}^t) \geq 2 \sum_{k\in K} (z_{jk}^{t-1} + z_{jk}^t) - 2 \qquad j \in J, t \in \mathcal{T} \setminus \{1\} \qquad (14)$$

$$\sum_{t\in\mathcal{T}} t z_i^t \leq \sum_{t\in\mathcal{T}} t z_j^t \qquad (i,j) \in P \qquad (15)$$

$$\sum_{t\in\mathcal{T}}\sum_{k\in K} \Delta_k v_{jk}^t = v_j^{max} \qquad j \in J \qquad (16)$$

$$a_j^t \geq a_i^t + \sum_{k\in K} v_{ik}^t + t_{ij} - \overline{T}(1 - \sum_{k\in K} x_{ijk}^t) \qquad (i,j) \in A, t \in \mathcal{T} \qquad (17)$$

$$a_0^t = 0 \qquad t \in \mathcal{T} \qquad (18)$$

$$0 \leq a_{n+1}^t \leq \overline{T} \qquad t \in \mathcal{T} \qquad (19)$$

$$0 \leq v_{jk}^t \leq \overline{T} z_{jk}^t \qquad j \in J, k \in K, t \in \mathcal{T} \qquad (20)$$

$$x_{ijk}^t \in \{0,1\} \qquad (i,j) \in A, k \in K, t \in \mathcal{T} \qquad (21)$$

$$z_{jk}^t \in \{0,1\} \qquad j \in J, k \in K, t \in \mathcal{T} \qquad (22)$$

$$z_j^t \in \{0,1\} \qquad j \in J, t \in \mathcal{T} \qquad (23)$$

$$g_j^l \in \{0,1\} \qquad j \in J, l = b_j^{min}, \ldots, b_j^{max}. \qquad (24)$$

Objective function (1) minimizes the total routing costs. Constraints (2) ensure that, on each day, each team is used at most once. Constraints (3) impose that each worker is assigned to at most one team on each day. Constraints (4) ensure that, on each day, at most D teams are used. Constraints (5) impose flow conservation. Constraints (6) guarantee that if job j is assigned to team k on day t, then this team has to visit the associated location. For each job, constraint (7) guarantees that its service starts in one of the days across the planning horizon. Constraints (8) select the number of visits to serve each job, that is, the number of consecutive days (ranging from the minimum b_j^{min} and the maximum b_j^{max}) that a team must visit the job's location to ensure its service is complete. Constraints (9) ensure that a job served over l days is assigned to exactly l teams across the time horizon. Constraints (10) forbid combinations of starting period t and durations l such that the job service would span beyond the length of the planning horizon. For instance, assume, for simplicity

that $T = 3$ and that job j must be served exactly across 2 days (i.e., $b_j^{min} = b_j^{max} = 2$). Constraint (10) allows its service to start on days 1 or 2, but not on 3. In this case $l + t = 2 + 3 > T + 1 = 3 + 1$, and the constraint turns out to be $z_j^{t=3} + g_j^{l=2} \leq 1$. Constraints (11) ensure that at most one team is assigned to each job on each day. Constraints (12) guarantee that a job started on day t is served by a suitable team on day t and in each of the following $b_j^{min} - 1$ days. In other words, it ensures that once a job is started, it is served for b_j^{min} consecutive days. Constraints (13) are used for the remaining $l - b_j^{min}$ periods. If job j started on day t is performed over a number l of days ranging from $b_j^{min} + 1$ to b_j^{max} (i.e., $\sum_{\bar{l}=l+1}^{b_j^{max}} g_j^{\bar{l}} = 1$), there must be a team serving the job on each day from $t + b_j^{min}$ to $t + l - 1$. Constraints (12) and (13) also ensure that jobs are performed over consecutive days. A job j must be the first job visited in a route on day t if it was started and not completed on a previous day, and the last job visited in a route in the previous day $t - 1$. This is imposed by constraints (14). Constraints (15) impose pairwise precedence constraints. Constraints (16) guarantee that the total service time performed on each job accounts for v_j^{max} units of time. The maximal daily work shift restriction is guaranteed through constraints (17)–(19). First, constraints (17) indicate the arrival time at job j from job i. If any team traverses arc $(i, j) \in A$ in day t (i.e., $\sum_{k \in K} x_{ijk}^t = 1$), the arrival time at job j is bounded by the arrival time at job i, plus the service time provided at job i and the travel time from i to j. If arc $(i, j) \in A$ is not traversed in period t by any team (i.e., $\sum_{k \in K} x_{ijk}^t = 0$), the inequality becomes ineffective by the subtraction of the term $\overline{T}$. Second, constraints (18) indicate that vehicles depart from the depot at time 0. Finally, constraints (19) guarantee that vehicles return to the depot no later than $\overline{T}$. Note that constraints (17)–(19) also eliminate subtours. Constraints (20) impose that the service time provided to each job is zero on those days where no team has been assigned to it. Finally, constraints (21)–(24) define the nature and domain of the decision variables.

3 Adaptive Large Neighborhood Search for the Multi-Period Workforce Scheduling and Routing Problem

In this paper, we develop a matheuristic based on the ALNS framework proposed by Røpke and Pisinger (2006). The ALNS is an extension of the Large Neighborhood Search (LNS) framework developed by Shaw (1998) that includes an adaptive layer. LNS is an example of very large-scale neighborhood search heuristics introduced by Ahuja et al. (2002). According to the authors, the latter is a family of heuristics that search neighborhoods whose size grows very rapidly with the problem size, or neighborhoods that are too large to be searched explicitly. The basic idea of LNS is to avoid an explicit definition of the neighborhood, which is rather defined implicitly by a *destroy* and a *repair* mechanisms. A destroy operator "destructs" part of a solution, which is later "reconstructed" by the repair operator. Hence, the neighborhood of a solution is implicitly defined as the set of solutions that can be reached by first applying a destroy operator followed by a repair operator. Their performance is recorded during the execution of the algorithm, so that the algorithm adapts to the current instance promoting the use of operators that have performed well.

The present section describes how the ALNS framework has been applied to the solution of the MPWSRP. A pseudocode of the ALNS is sketched in Algorithm 1. The main input of the ALNS is an initial feasible solution s. This solution is generated by means of the construction heuristic described in Section 3.1. The destroy operators that are used in our ALNS are described in Section 3.2, whereas the repair operators are detailed in Section 3.3. We have defined two classes of destroy operators: one class removes jobs (according to different criteria), whereas the other destroys teams of workers. Consequently, we have designed two classes of repair operators: one class reinserts jobs,

whereas the other reconstructs teams. In a standard ALNS, the selection of which pair of destroy and repair operators to apply in each iteration is done independently, based on their respective weights. Conversely, in our ALNS we need to guarantee that, in each iteration, if a given destroy operator is applied, then one among the corresponding repair operators must be employed (e.g., if a method that removes jobs is selected, then a method that reinsert jobs must be chosen). To this aim, we construct a priori *suitable pairs* (d, r) *of destroy-repair operators*, and assign a weight ω_{dr} and a score ψ_{dr} to each of such pairs. Hence, in every iteration of ALNS a destroy-repair operator pair (d, r) is chosen using a roulette-wheel selection based on the performance recorded in previous iterations, i.e., weights ω_{dr} (line 3). This selection principle is described in details in Section 3.4. Subsequently, the heuristic first applies the chosen destroy operator d to the current solution s, which is partially destroyed obtaining solution s' (line 4). Then, the chosen repair operator r is applied to the partially destroyed solution s', returning a new solution s^{new} (line 5). Note that, given the complexity of the MPWSRP, it is sometimes hard for the proposed repair operators to return a feasible solution in each iteration. Hence, we allow the repair operators to return an infeasible solution, which is penalized by increasing the objective function value by 15%. Solution s^{new} is then evaluated to determine if it should become the new current solution, or it should be rejected (lines 6–14). In our implementation, we follow Røpke and Pisinger (2006) and use a simulated annealing acceptance criterion where the new solution s^{new} is always accepted if $z(s^{new}) < z(s^*)$, where $z(s)$ is the objective function value of solution s. In this case, a new best-known solution s^* is found (line 7), and the composition of each team is reoptimized by invoking, first, the ALL_TEAMS_REMOVAL operator and, then, the TEAM_REPAIR operator (see in the following for further details). On the other hand, if $z(s^{new}) \geq z(s^*)$, the counter it_{noImpr} of the iteration without an improvement is increased (line 10), and solution s^{new} is accepted with probability $e^{-(z(s^{new})-z(s))/\tau}$ (lines 11–12), where $\tau > 0$ is the current *temperature*. The temperature τ is initialized at τ_{start} and is decreased at each iteration as $\tau = c\tau$, where $0 < c < 1$ is the *cooling rate*. The algorithm proceeds by updating the score associated with the destroy-repair operator pair (d, r) chosen (line 15). At every $iter_{upd}$ iterations, the weight $\omega_{(d,r)}$ associated with each destroy-repair operator pair is updated (line 17), whereas the scores are reinitialized to zero (line 18). The ALNS stops when $iter_{max}$ iterations have been performed or $iter_{maxNoImpr}$ consecutive iterations without improvement have been carried out (line 21).

3.1 Initial Solution Procedure

This section describes the construction heuristic employed to generate an initial feasible solution s used as input for the ALNS. In general terms, the procedure schedules one day at a time, using a two-phase approach. In the first phase, a set of reference jobs (henceforth called *seed jobs*) is identified and a team is constructed for each of them. In the second phase, more jobs are possibly assigned to the teams constructed. We now detail the operation of each of these two phases, which are both executed each day, starting from $t = 1$, until all jobs have been served completely.

PHASE ONE: Find seed jobs.
The objective of the first phase is to identify, on each day, a set of seed jobs among those not yet completed. The basic idea is to find a set of *heterogeneous*, in some sense, jobs and then associate a different team to each of them. If the seed jobs are "sufficiently heterogeneous" the teams are, consequently, dissimilar from each other. This should facilitate assigning further jobs in the second phase, or, at least, reduce the chances of encountering an infeasible assignment because no team able to serve a job has been created in this first phase. Note that, as in the MPWSRP we are given a limited fleet of D vehicles, each day at most D seed jobs can be determined. Furthermore, to respect

Algorithm 1 General Scheme of the ALNS.

Input: A feasible solution s.
Output: The best-known solution s^*.

1. Set $it = 0$, $it_{noImpr} = 0$, $s^* = s$, $\omega = (1, \ldots, 1)$, and $\psi = (0, \ldots, 0)$.
2. **repeat**
3. Select a destroy-repair operator pair (d, r) by roulette-wheel selection using weights ω.
4. Apply $d(s)$, obtaining s'.
5. Apply $r(s')$, obtaining s^{new}.
6. **if** $z(s^{new}) < z(s^*)$ **then**
7. Set $s^* = s'$, $s = s^{new}$, and $it_{noImpr} = 0$.
8. Reoptimize team compositions.
9. **else**
10. $it_{noImpr} = it_{noImpr} + 1$.
11. **if** s^{new} satisfies the acceptance criterion **then**
12. Set $s = s^{new}$.
13. **end if**
14. **end if**
15. Update score ψ_{dr}.
16. **if** $it \,\%\, iter_{upd} = 0$ **then**
17. Update weights ω.
18. Set scores $\psi = (0, \ldots, 0)$.
19. **end if**
20. $it = it + 1$.
21. **until** $it = iter_{max}$ **or** $it_{noImpr} = iter_{maxNoImpr}$

priorities, any job with a given priority can become a seed only if all jobs with a higher priority have been started on a previous day, or have been already selected as seed jobs for the current day.

Seed jobs are identified based on a linear weighted combination of the following four criteria:

1. *criticality* (α_j for job j, $j = 1, \ldots, n$): it measures how important (i.e., critical) is to serve job j the earliest. For each job j not yet started, it is equal to its service time v_j^{max}. It is worth noting that, between two jobs having the same priority, the one with the largest service time is the most critical. To guarantee that if a job service spans across multiple days the latter must be consecutive, the criticality of each job j whose service started on a previous day and is not completed yet is set to a very large value (i.e., $\alpha_i := +\infty$);

2. *difficulty* (β_j for job j, $j = 1, \ldots, n$): it measures how difficult is to serve job j. It is computed as the sum over the different skill domains required to serve the job (i.e., $\beta_j := \sum_{s=1}^{S} r_{js}$);

3. *similarity* ($\gamma_{j\mathcal{S}}$ for job j, $j = 1, \ldots, n$): it measures how similar is job j, in terms of skills required, compared with the other seed jobs already determined for the current day. Note that the more similar, the worse. For job j, it is computed as $\gamma_{j\mathcal{S}} := \sum_{i \in \mathcal{S}} \gamma_{ji}$, where $\mathcal{S}$ is the set of seed jobs already determined for the current day, and γ_{ji} is the similarity between jobs j and i. The latter is computed as $\gamma_{ji} := \sum_{s=1}^{S} |r_{js} - r_{is}|$. When computing the similarity for the first seed job on each day, $\gamma_{j\mathcal{S}}$ is by definition set equal to zero;

4. *distance* (δ_j for job j, $j = 1, \ldots, n$): it measures the traveling cost from the depot (vertex 0) to the job location (vertex j), computed as $\delta_j := c_{0j}$. Note that the larger the distance, the worse.

A *score* $SSeed(j)$ for each job j not yet completed is computed weighting the former four criteria as follows: $SSeed(j) := \pi_\alpha \alpha_j + \pi_\beta \beta_j - \pi_\gamma \gamma_{j\mathcal{S}} - \pi_\delta \delta_j$. As mentioned above, $SSeed(j)$ is computed for each job whose service is not completed yet. The job that yields the highest score, say job j', is selected as a potential seed job. Before selecting job j' as the next seed job, the procedure determines if it is possible to create a team with the available workers (i.e., the unassigned workers) that can serve it. If it is possible to create a team, then job j' is declared to be the next seed job, and the corresponding team is created using the greedy heuristic CONSTRUCT_TEAM detailed below. Conversely, if on the current day it is not possible to create a team to serve job j' with the unassigned workers, this job is temporarily removed from the list of jobs not yet completed (it will be added to the list of the next day), and the following best job, based on its score, is evaluated.

Once the next seed job has been determined, the CONSTRUCT_TEAM heuristic is invoked to create a team that can serve it. The heuristic iteratively assigns an unassigned worker to the team, until all skills needed to serve the seed job have been covered by at least one worker, and the minimum number of workers is reached. The heuristic always assigns first the worker that covers the largest number of required skills that are not covered yet. Ties are broken choosing the worker with the minimum number of skills not required to serve the seed job as these are, in some sense, skills wasted (see below for further details). Any further tie is broken choosing the unassigned worker with the smallest index.

Each time a team is created, the algorithm checks the number of unassigned workers on the current day. If this number is smaller than $\theta|W|$, the procedure searching for further seed jobs is stopped. The value of $\theta \in [0, 1]$ indicates a sort of safety level of workers. It has been introduced to leave some workers unassigned at the end of PHASE ONE, which can be added to the existing teams in the following PHASE TWO. Indeed, in the second phase new jobs will be assigned to the existing teams. These unassigned workers can then be used to adjust an existing team when it lacks of some skills required to serve a new job.

PHASE ONE continues looking for new potential seed jobs for the current day until one of the following conditions is encountered:

1. D seed jobs have been determined;
2. the number of unassigned workers is smaller than the safety level;
3. all jobs have been entirely assigned.

Recall that the actual service time to serve a job depends on the number of workers composing the team that has been assigned to it. Particularly, the more workers, the better. Therefore, it is always convenient to have large teams, as this allows a team to serve, hopefully, several jobs on a day before returning to the depot. If after identifying the set of seed jobs for the current day, the number of unassigned workers is larger than the safety level, the algorithm tries to add unassigned workers to the teams created. Unassigned workers are added to teams in a greedy fashion. All possible assignments of an unassigned worker, say w, to the teams available are evaluated. For each possible assignment, say worker w to team k, the procedure computes the number of skill domains in which worker w is proficient and that are already covered by other workers in team k. Note that these skills are, in some sense, *wasted*, as they are covered already by other team members. Let W_k be the set of workers already assigned to team k. The number of skills wasted if worker w is assigned to team k is computed as $\sum_{s=1}^{S}(1 - |p_{ws} - \max_{w' \in W_k} p_{w's}|)$. (Recall that p_{ws} is a binary parameter.) Once all possible assignments have been evaluated, the one that minimizes the number of skills wasted is implemented (ties are broken choosing the unassigned worker with the smallest

index), and the actual service times of all jobs served by the corresponding team are updated. Other unassigned workers are iteratively added to the existing teams as long as their number is larger than the safety level.

Note that, for a given day t, the main outcome of PHASE ONE is a set of at most D teams, each one corresponding to a route r_t that visits exactly one seed job. PHASE TWO is then applied, as detailed below.

PHASE TWO: Assign additional jobs.
In PHASE TWO, the procedure tries to assign any unassigned job to the teams created in the former phase. Similar to PHASE ONE, in order to respect priorities, any unassigned job can become a candidate for insertion only if all jobs with a higher priority have been already started.

In this phase, the algorithm computes the score to insert each candidate job, say j, into each route r_t available for the current day t, in its cheapest and feasible position. Similar to seed jobs, the score is computed as a linear weighted combination of the following criteria:

1. *criticality* (α_j, for job j, $j = 1, \ldots, n$): as for the seed jobs, it is equal to its service time v_j^{max};

2. *difficulty* (β_{jr_t}, for job j, $j = 1, \ldots, n$, and route r_t used on day t): it is computed as the additional number of workers needed to cover the skill domains not already covered by the team currently associated with route r_t;

3. *insertion cost* (δ_{jr_t}, for job j, $j = 1, \ldots, n$, and route r_t used on day t): it is the additional traveling cost incurred inserting job j into route r_t in its best feasible position identified using a cheapest insertion routine;

4. *reduce unnecessary job splitting* (ϕ_{jr_t}, for job j, $j = 1, \ldots, n$, and route r_t used on day t): it is used to penalize situations where inserting job j into route r_t makes its remaining service to be split across multiple days. Particularly, it takes value zero if after inserting job j into route r_t, the job is entirely served, or when the service must span across multiple days since the remaining service time is larger than $\overline{T} - 2t_{0j}$. On the other side, if the remaining service time is smaller or equal than $\overline{T} - 2t_{0j}$, it is equal to the service time remaining after inserting job j into route r_t;

5. *limit the delay of jobs with smaller priorities* (ρ_{jt}, for job j, $j = 1, \ldots, n$ and day t): it is computed as $\rho_{jt} = \max\{0, t - \max_{j' \in J' | p_j = p_{j'}}\{a_{j'}\}\}$, where J' refers to the jobs currently served, even if only partially. $a_{j'}$ represents the day when job j' is started in the current solution, and the expression $\max_{j' \in J' | p_j = p_{j'}}\{a_{j'}\}$ refers to the latest starting day among all jobs j' currently assigned and that have the same priority as j. This term is inserted to limit the impact on the start of jobs with a smaller priority than j. Recall that to start a job with a given priority, all jobs with higher priorities have to be started. Hence, delaying the start of job j might also delay the start of the unassigned jobs with smaller priorities. Therefore, if inserting job j in any route on day t does not delay the starting of the currently assigned jobs with the same priority, then ρ_{jt} is set equal to zero. Otherwise, it is computed as the delay (expressed in days) due to the insertion.

Once all the previous criteria have been computed, a score $SJob(j, r_t)$ is determined applying the following weighted linear function: $SJob(j, r_t) := \hat{\pi}_\alpha \alpha_j - \hat{\pi}_\beta \beta_{jr_t} - \hat{\pi}_\delta \delta_{jr_t} - \hat{\pi}_\phi \phi_{jr_t} - \hat{\pi}_\rho \rho_{jt}$. If the insertion of job j into route r_t is not feasible, we set $SJob(j, r_t) = -\infty$. The next job to be inserted is the one that yields the highest score, say job j' to be inserted into route r'_t in its cheapest feasible

position. Then, the procedure adds to team k' associated with route r'_t, if necessary, the additional workers needed to serve job j' by invoking procedure CONSTRUCT_TEAM already described. PHASE TWO stops when no further feasible insertion is available on the current day, or when all jobs have been entirely assigned.

PHASE ONE and PHASE TWO are iteratively repeated for each day, until all jobs have been entirely assigned. At that point, the procedure stops returning an initial feasible solution s which is improved using the ALNS algorithm described in the following.

3.2 Destroy Operators

Several different destroy operators have been previously proposed in the literature. In the following, we describe those employed here, which are adaptations to the MPWSRP of well-known operators. As mentioned above, we have defined two classes of destroy operators: one class that removes jobs, whereas the other destroys teams of workers. These are described next.

DESTROY JOBS OPERATORS

All the following operators take as input the current solution s and an integer λ_J. The output of each procedure is a partly destroyed solution s' where λ_J jobs have been removed. Before invoking any of the following destroy operators, the algorithm determines the value of λ_J. This number is determined using a semi-triangular distribution with negative slope, which favors the removal of a small number of jobs. In fact, with a semi-triangular distribution small random numbers appear with high probabilities, whereas large numbers appear with low probabilities. More specifically, λ_J is an integer random number drawn from the interval $[1, n_{max}]$, where $n_{max} = \lfloor 0.30 \times |J_s| \rfloor$, and $|J_s|$ is the number of jobs served completely in solution s. λ_J is drawn according to the formula $\lambda_J = \left\lfloor n_{max} - \sqrt{(1-u)(n_{max}-1)^2} \right\rfloor$, where u denotes a uniform random number drawn in the interval $[0, 1]$.

RANDOM_JOB_REMOVAL

This is the simplest destroy operator. It randomly selects λ_J jobs and removes them from the current solution. It is worth highlighting that if job j is served across multiple days, this operator removes the service to j on each of these days.

WORST_JOB_REMOVAL

This operator aims at removing jobs with the worst impact on the objective function value, and is based on the akin operator developed in Røpke and Pisinger (2006).

Given a job j served in the current solution s, possibly across multiple days, we denote as $z(s)$ the objective function value of solution s. If job j is served completely on one single day, let $z(s \setminus j)$ be the objective function value of solution s after removing job j and replacing the removed arcs with the arc incident to its predecessor and successor. On the other side, if job j is served across multiple days, let $z(s \setminus j)$ be the objective function value of solution s after removing job j on every route it is served. Note that this choice favors the removal of jobs that are served across multiple days, since these jobs tend to yield the highest impact on the objective function value. We denote by $\Lambda(j, s) = z(s) - z(s \setminus j)$ the cost of job j. The WORST_JOB_REMOVAL operator removes λ_J jobs from the current solution with high values of $\Lambda(j, s)$. The job selection is randomized, but is arranged in such a way that jobs with high values of $\Lambda(j, s)$ are most likely chosen. In more details, an ordered list L is created sorting all jobs in non-increasing order of $\Lambda(j, s)$. Then, a random number r uniformly distributed in $[0, 1]$ is drawn, and the job in position $|L| r^{p_{worst}}$ in list L is removed, where p_{worst} is a

given parameter that controls the degree of randomization.

RELATED_JOBS_REMOVAL

This operator, often called in the literature the Shaw removal operator since its first proposal by Shaw (1998), aims at removing jobs that are, to a certain extent, similar. We denote the relatedness between two jobs i and j by rel_{ij}. This measure is computed as a linear weighted combination of the following six terms:

1. c_{ij}: travel cost associated with arc (i, j) and measures the relatedness in terms of distance;
2. t_{ij}: travel time associated with arc (i, j) and measures the relatedness in terms of time;
3. γ_{ij}: similarity between jobs i and j in terms of skills required, computed as described above for PHASE ONE;
4. $|p_i - p_j|$: similarity between jobs i and j in terms of their priority level ;
5. $|y_i^{min} - y_j^{min}|$: similarity between jobs i and j in terms of the minimum number of workers required to perform them;
6. $|a_i - a_j|$: where a_i is, as above, the day that job i is started in the current solution s. It measures the similarity in terms of starting day of the two jobs in solution s.

The relatedness rel_{ij} is computed weighting the former terms as follows: $rel_{ij} := \bar{\pi}_d c_{ij} + \bar{\pi}_t t_{ij} + \bar{\pi}_\gamma \gamma_{ij} + \bar{\pi}_p |p_i - p_j| + \bar{\pi}_w |y_i^{min} - y_j^{min}| + \bar{\pi}_a |a_i - a_j|$. Note that the smaller rel_{ij}, the more related the two jobs.

Let R be the set of jobs to remove, initially empty. This operator initially selects randomly one job to remove, say job i. Set R is then initialized as $R = \{i\}$. In the following iterations, the procedure selects the other $\lambda_J - 1$ jobs to remove as follows. A job i' from set R is randomly selected. An ordered list L' is created sorting all jobs in $J \backslash R$ in non-decreasing order of their relatedness with job i' (that is, from job j with the smallest value $rel_{i'j}$ to job j' with the largest value $rel_{i'j'}$). Then, a random number r' uniformly distributed in $[0, 1]$ is drawn, and the job in position $|L'|r'^{p_{rel}}$ in list L' is removed, where p_{rel} is a given parameter that controls the determinism in the selection of the jobs (the larger p_{rel}, the more likely jobs similar to those already in R are selected).

DESTROY TEAMS OPERATORS

The following two operators take as input the current solution s and return a partly destroyed solution s' where a number of teams is completely destroyed.

RANDOM_TEAM_REMOVAL

This operator requires, as an additional input, an integer number λ_K. The output of this procedure is a solution where λ_K teams, randomly selected, have been removed from the current solution s. Note that, removing a team implies unassigning all workers composing it. Akin to the operators that remove jobs, the value of λ_K is determined using a semi-triangular distribution with negative slope, which favors the removal of a small number of teams. Let $|K_s|$ be the number of teams used in solution s. Then, λ_K is drawn from the interval $[1, |K_s|]$ according to the formula $\lambda_K = \left\lfloor |K_s| - \sqrt{(1-u)(|K_s|-1)^2} \right\rfloor$, where u is defined as above.

ALL_TEAMS_REMOVAL

This operator destroys all teams used in the current solution s.

3.3 Repair Operators

In order to repair a solution s' partly destroyed by applying one of the destroy operators, we have designed several repair operators. As mentioned, we have developed two classes of these operators: one class that reinserts jobs, whereas the other reconstructs teams. The two classes of operators are described in the following.

REPAIR JOBS OPERATORS
All the following operators take as input a partial solution s' where a number of jobs have been removed. The output of each procedure is a, possibly infeasible, solution s^{new}.

GREEDY_JOB_INSERTION
Let J_u be the set of the jobs currently unassigned. This operator iteratively inserts a job from J_u into a route (or routes, if the service of the job spans across multiple days) at its best feasible position. Similarly to PHASE TWO of the initial solution procedure, the job to insert at each iteration is identified based on its score. Additionally, any unassigned job can become a candidate for insertion only if all jobs with a higher priority have been already started.

The score of each job j is computed as follows. If job j can be served in one day (i.e., $b_j^{min} = 1$), the algorithm first analyzes each day composing the planning horizon, one at a time, looking for feasible insertions such that the job is served completely in one day. More specifically, for each day t, a score is computed for inserting job j in each route r_t available, i.e., each route used on day t in solution s' (in this case, the job is inserted in its cheapest feasible position), as well as in any empty route if not all the D vehicles are used and it is possible to create a team serving the job with the currently unassigned workers. In this phase, any insertion of job j into a route is infeasible whenever it does not allow to completely serve the job. It is also infeasible to insert job j if this violates the priority requirement. This might happen when, in the partly destroyed solution, a job i with a smaller priority, say $p_i > p_j$, is served on day $t-1$. In this case, job j cannot be inserted later than $t-1$. The score for each of these jobs is computed as $SJob_1(j, r_t) := \hat{\pi}_\alpha \alpha_j - \hat{\pi}_\delta \delta_{jr_t} - \hat{\pi}_\rho \rho_{jt}$, where all terms are computed as described above for PHASE TWO. The move yielding the highest score $SJob_1(j, r_t)$ represents the best feasible insertion of job j that guarantees its service in one single day.

If the job can be served across multiple days (i.e., $b_j^{max} > 1$), identifying its best feasible insertion is considerably more complicated. In fact, the constraint that the job must be served over consecutive days implies that any feasible insertion has to comply with the following requirements. Assuming that all the other problem constraints are satisfied (such as, priorities and team feasibility), Figure 1 provides an example of the feasible and infeasible insertions of job j served across 4 days. Note that it is possible to identify three cases: the insertion concerns the *first day* (the service of job j starts); the insertion concerns the *last day* (the service of job j ends); the insertion pertains one of the *middle days* (that is, between the start and the end of the service of job j). In fact, to guarantee that if a team starts working on job j on a given day, then the job must be continually carried out until finished, the only feasible insertion on the first day into an existing route is as the last vertex visited before the depot, or into a new route. On a middle day, job j can only be inserted into a new route, whereas on the last day it can be inserted into a new route, or into an existing route only as the first vertex visited after the depot.

The traditional approach would be to enumerate and evaluate all feasible insertions. Consider the following example: let us assume that the length T of the planning horizon is 5, the number D of vehicles available is 5, and the job j to be inserted must be served exactly across two days (i.e., $b_j^{min} = b_j^{max} = 2$). Figure 2 shows a partly destroyed solution s' where exactly 5 teams are used on

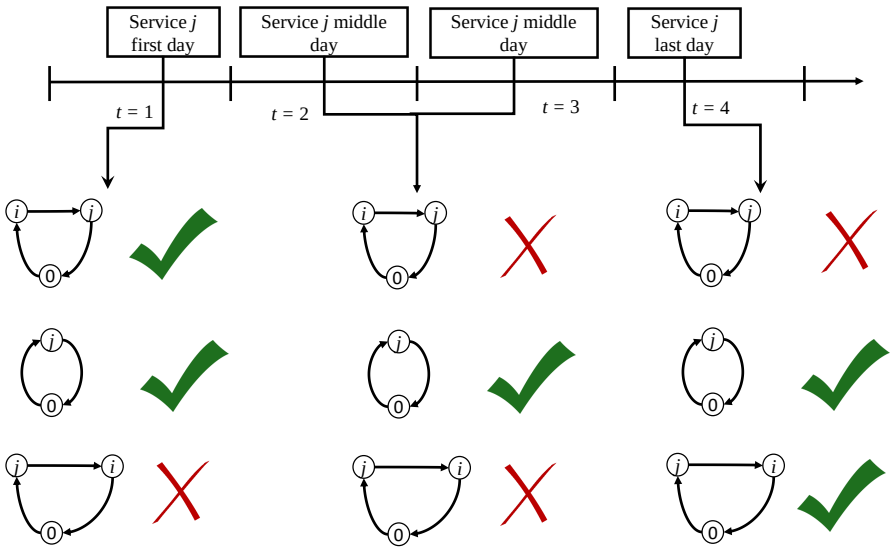

Figure 1: An illustrative example of the feasible (green checkmarks) and infeasible (red crosses) insertions of a job j served across 4 days.

each day. To ease of readability, each team serves in s' exactly one job, and routes used on days $t = 3$ and $t = 4$ are not displayed. Let us assume that it is feasible to insert job j into each route, provided it is inserted as the last vertex in the route on its first day, and as the first vertex in the route on its last (i.e., second) day. Figure 2 highlights with a dashed arrow the insertion of job j into the first route on day 1 as its first day. This insertion must be evaluated in combination with the insertion of job j in each route on day 2 as its last day. Altogether, in this example the total number of feasible insertions to be consider is equal to 5 (routes available as first day insertion) $\times$ 5 (routes available as last day insertion) $\times$ 4 (number of pairs of consecutive days), totaling 100 feasible insertions. Note that, for each of these pairs of insertions, one also has to be determine the service time provided on each day the job is visited.

In preliminary experiments, we found that it is more efficient to solve an optimization model, rather than using the traditional approach exemplified above. To ease of readability, in the following description we have suppressed the subscript j, but all sets and decision variables have to be intended as concerning job $j \in J_u$. For each day $t \in \mathcal{T}$, the operator creates set $M_t = M_t^{fi} \cup M_t^{mi} \cup M_t^{la}$, which contains all feasible insertions that can be implemented on day t. This set is given by the union of set M_t^{fi}, which contains all feasible insertions of job j on day t as first day; set M_t^{mi}, which comprises all feasible insertions on day t as middle day; and set M_t^{la}, which contains all feasible insertions on day t as last day. Note that, by definition, set M_t^{fi} is empty for all $t = T - b_j^{min} + 2$ (i.e., job j can, at the latest, start day $T - b_j^{min} + 1$). Analogously, set $M_t^{la} = \emptyset$ for day $t = 1$. Furthermore, note that if, for a given day t, there are two or more feasible insertions as a middle day, these moves are interchangeable. Hence, to limit the number of binary variables used in the optimization model, for each day $t = 1, \ldots, T$ set M_t^{mi} contains at most one feasible insertion, the one corresponding to the unused vehicle with the smallest index. For each insertion move $m \in M_t$, with $t = 1, \ldots, T$, the algorithm determines the insertion cost δ_m, and the service time v_m that can be provided. Furthermore, for each period t it computes the value ρ_t of the delay introduced if the service of job j starts on day t. Note that in the example of Figure 2 this approach requires the evaluation of 25 possible insertions (one evaluation per each route available on each day). Furthermore, the service

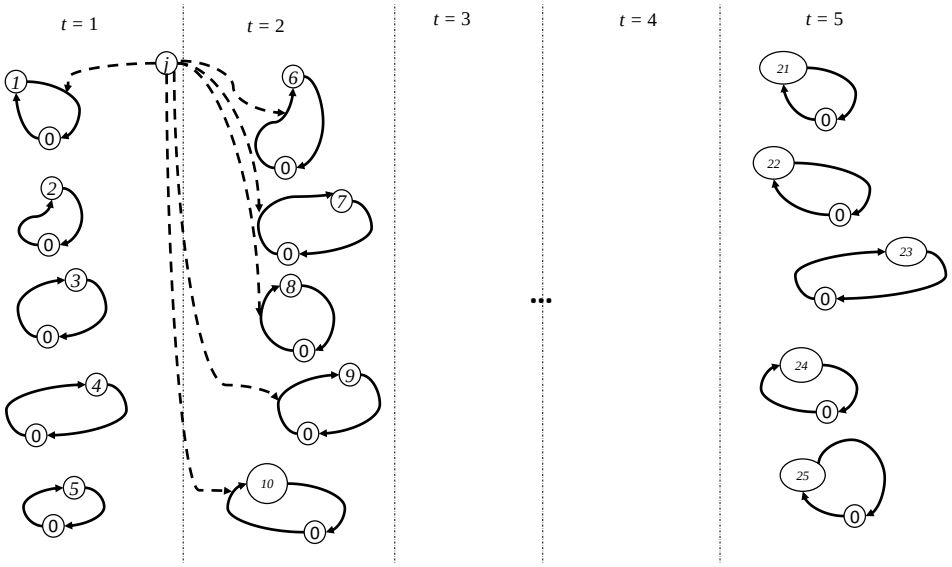

Figure 2: An example showing the feasible insertions of job j.

time to provide on each day is determined by solving the optimization model.

The optimization model uses the following variables. Let $z_m^t \in \{0,1\}$ take value 1 if insertion $m \in M_t$, with $t \in \mathcal{T}$, is chosen, and 0 otherwise. Let $g^l \in \{0,1\}$ take value 1 if job j is served across l consecutive days, with $l = b_j^{min}, \ldots, b_j^{max}$, and 0 otherwise. Finally, let $z^t \in \{0,1\}$ take value 1 if the work on job j starts on day t, with $t \in \mathcal{T}$, and 0 otherwise. The problem of identifying the best feasible insertion of an unassigned job $j \in J_u$ can be cast as the following ILP model:

$$\min \sum_{t \in \mathcal{T}} \left(\sum_{m \in M_t} \hat{\pi}_\delta \delta_m z_m^t + \hat{\pi}_\rho \rho_t z^t \right) \tag{25}$$

$$\text{s.t.} \sum_{t \in \mathcal{T}} \sum_{m \in M_t} v_m z_m^t \geq v_j^{max} \tag{26}$$

$$\sum_{m \in M_t} z_m^t \leq 1 \qquad t \in \mathcal{T} \tag{27}$$

$$\sum_{l=b_j^{min}}^{b_j^{max}} g^l = 1 \tag{28}$$

$$\sum_{t \in \mathcal{T}} \sum_{m \in M_t} z_m^t = \sum_{l=b_j^{min}}^{b_j^{max}} l g^l \tag{29}$$

$$\sum_{t=1}^{T-b_j^{min}+1} z^t = 1 \tag{30}$$

$$\sum_{m \in M_t^{fi}} z_m^t = z^t \qquad t = 1, \ldots, T - b_j^{min} + 1 \tag{31}$$

$$\sum_{m \in M_t^{mi}} z_m^{t+l} \geq z^t \qquad l = 1, \ldots, b_j^{min} - 2, t \in \mathcal{T} \tag{32}$$

$$\sum_{m \in M_t^{mi} \cup M_t^{la}} z_m^{t+b_j^{min}-1} \geq z^t \qquad t \in \mathcal{T} \tag{33}$$

$$\sum_{m \in M_t^{mi} \cup M_t^{la}} z_m^{t+l} \geq z^t + \sum_{\bar{l}=l+1}^{b_j^{max}} g^{\bar{l}} - 1 \qquad l = b_j^{min}, \ldots, b_j^{max} - 1, t \in \mathcal{T} \tag{34}$$

$$\sum_{m \in M_t^{la}} z_m^{t+l} \geq z^t + g^{l+1} - 1 \qquad l = b_j^{min} - 1, \ldots, b_j^{max} - 1, t \in \mathcal{T} \tag{35}$$

$$\sum_{t \in \mathcal{T}} \sum_{m \in M_t^{la}} z_m^t = 1 \tag{36}$$

$$z^t + \sum_{l=b_j^{min} | l+t > T+1}^{b_j^{max}} g^l \leq 1 \qquad t \in \mathcal{T} \tag{37}$$

$$z_m^t \in \{0,1\} \qquad t \in \mathcal{T}, m \in M_t \tag{38}$$

$$g^l \in \{0,1\} \qquad l = b_j^{min}, \ldots, b_j^{max} \tag{39}$$

$$z^t \in \{0,1\} \qquad t \in \mathcal{T}. \tag{40}$$

Objective function (25) minimizes the sum of two weighted components. The first term represents the insertion cost, whereas the second term measures the delay introduced by starting job j on day t. Constraint (26) ensures that the total service time provided is not smaller than v_j^{max}. For each day t, constraint (27) guarantees that at most one insertion move is chosen. Constraint (28) along with (29) state that exactly l insertion moves, with $l = b_j^{min}, \ldots, b_j^{max}$, must be selected. Constraint (30) establishes that the service on job j starts on one of the feasible days (i.e., between day 1 and day $T - b_j^{min} + 1$). Constraints (31)–(35) guarantee that job j is continually served across consecutive days, on one side, and that, depending on the day the job service starts, the correct type of insertion move is chosen. More specifically, let us assume that the work of job j starts on day t (i.e., $z^t = 1$). Then, constraint (31) ensures that one of the feasible insertions on day t as first day is chosen. Constraints (32) guarantee that in each of the following $b_j^{min} - 2$ days one of the feasible insertions as a middle day is chosen. Constraint (33) establishes that in day $t + b_j^{min} - 1$ one of the feasible insertions as a middle or last day is selected. Constraints (34) ensure that if job j is served across a number l of days between b_j^{min} and $b_j^{max} - 1$, one feasible insertion move as a middle or last day must be chosen for each day from $t + b_j^{min}$ and $t + l$. Constraints (35) enforce that if job j is served across l days, then on day $t + l - 1$ one of the feasible insertions as last day is chosen. Constraint (36) imposes than only one feasible insertion as a last day is selected in any feasible solution. Constraints (37) forbid combinations of starting period t and durations l such that the job service would span beyond the length of the planning horizon. Finally, constraints (38)–(40) define the nature and domain of the decision variables.

An optimal solution to model (25)–(40) defines the best feasible insertion of job j across multiple days. The corresponding score is computed as $SJob_2(j) := \hat{\pi}_\alpha \alpha_j - \hat{\pi}_\delta \hat{\delta}_j - \hat{\pi}_\rho \hat{\rho}_j$, where the first term is computed as described above for Phase Two. In the second term, $\hat{\delta}_j$ is computed as the sum of the values of δ_m of each insertion move m selected in the optimal solution. Finally, in the third term, $\hat{\rho}_j$ corresponds to the value ρ_t related to the day a team starts serving job j in the optimal

solution. For each job that can be served on a single day (i.e., $b_j^{min} = 1$) and also on multiple days (i.e., $b_j^{max} > 1$), the algorithm, firstly, determines the insertion with the best score $SJob_1$ and, then, the insertion with the best score $SJob_2$. The insertion having the highest score, between these two is chosen as the best move. Let $SJob_{1,2}(j)$ denote the best feasible insertion for job $j \in J_u$ determined with this procedure.

Once the best feasible insertion of each unassigned job in set J_u has been determined, the job having the highest score is selected, and inserted into the partly destroyed solution s'.

Regret_Job_Insertion

This operator is based on the Greedy_Job_Insertion one described above, integrating a look-ahead information, called the *regret value*, while selecting the next job to insert. Intuitively speaking, in its simplest version the regret value measures the difference, in terms of score, between implementing the best insertion of job j and the second-best insertion. The operator computes the regret value ν_j for each unassigned job $j \in J_u$ as follows. Consider an unassigned job j, and let $SJob_{1,2}^{\kappa}(j, r_t)$ be the score related to its κ-best insertion. To determine the latter score, the procedure described above for the Greedy_Job_Insertion operator is iteratively performed until the κ-best insertion has been determined. Note that for each job that can be served across multiple days, we solve model (25)–(40) iteratively. In these cases, every time that at a given iteration the best insertion is determined from the solution of the latter optimization model, all variables corresponding to the optimal insertion moves are removed from the model in the subsequent iterations. The regret value for job j is computed as $\nu_j = \sum_{\kappa'=2}^{\kappa}(SJob_{1,2}^{1}(j, r_t) - SJob_{1,2}^{\kappa'}(j, r_t))$.

At each iteration, the Regret_Job_Insertion operator chooses the job that maximizes its regret value ν_j. If for some jobs it is not possible to identify at least κ feasible insertions, then the job with the fewest number of feasible insertions is chosen. Ties are broken choosing the job whose best insertion has the highest score. In our experiments, we used four values of κ: 2, 3, 4, and 5.

Objective Function Randomization

As recommended by Røpke and Pisinger (2006), we randomize the repair jobs operators described above by applying a noise term to the objective function value of the original problem. More specifically, our ALNS uses two versions of each of the above-described repair operators: one deterministic version, and another one that applies a random noise to the insertion costs (i.e., δ_{jr_t} and δ_m). With the latter version of the operators, every time that an insertion cost is calculated, the algorithm also computes a random noise value in the interval $\left[-\eta \times \max_{(i,j)\in A} c_{ij}, \eta \times \max_{(i,j)\in A} c_{ij}\right]$, which is then added to the insertion cost. Parameter η controls the amount of noise introduced.

REPAIR TEAMS OPERATORS

The following operator takes as input a solution s' partly destroyed by removing a number of teams, and provides as output a, possibly infeasible, solution s^{new}.

Team_Repair

This operator constructs a feasible team from scratch for each route that, in solution s', is not associated with a team, and tries to extend each of the other teams by adding unassigned workers if less than Q workers are currently assigned. Recall that it is always beneficial to have as large teams as possible, as this reduces the actual service times. Note that shorter service times for a job can potentially lead to fewer visits to a given job, and, consequently, lower traveling costs. This operator simultaneously constructs and adjusts teams by solving the optimization model described below. Conducting some preliminary experiments, we found that this approach often yields solutions

with shorter service times compared to the greedy heuristic CONSTRUCT_TEAM used in the initial solution procedure, at the expense of slightly longer computing times.

With a little abuse of notation, in the following we denote with the index $k \in K$ both a route and the team associated with it (to be constructed or already available from solution s'). Furthermore, let W_u^t denote the set comprising all unassigned workers available on day t. Let h_j be a parameter representing a penalty for serving job j across more than b_j^{min} days. The optimization model also uses the 0-1 parameter e_{ijk}^t, which takes value 1 if and only arc $(i,j) \in A$ is traversed in route $k \in K$ on day t in solution s', and the 0-1 parameter e_{wk}^t which takes value 1 if and only if worker $w \in W \backslash W_u^t$ is assigned to team $k \in K$ on day t in solution s'.

The mathematical formulation uses the following decision variables. Let $y_{wk}^t \in \{0,1\}$ take value 1 if worker $w \in W_u^t$ is assigned to team $k \in K$ on day t, and 0 otherwise. Let z_{jk}^t take value 1 if team $k \in K$ serves job $j \in J$ on day t, and 0 otherwise. In the optimization model, the value of variable z_{jk}^t is set according to solution s', but giving the chance of reducing the number of visits to job j by not serving it on the last day it is currently served in s', if this is feasible. To this aim, let t'_j be the last day job j is served by any team in solution s'. Let z_{jkq}^t take value 1 if team $k \in K$, which serves job $j \in J$ on day t, is made up of q workers, and 0 otherwise. Finally, the meaning of the remaining decision variables required by the model is akin to that described in Section 2. The mathematical programming formulation used by operator TEAM_REPAIR is as follows:

$$\min \sum_{j \in J} \sum_{k \in K} \left[\sum_{t \in \mathcal{T}} v_{jk}^t + h_j \left(\sum_{t \in \mathcal{T}} z_{jk}^t - b_j^{min} \right) \right] \tag{41}$$

$$\sum_{j \in J} v_{jk}^t \leq \overline{T} - \sum_{(i,j) \in A} e_{ijk}^t t_{ij} \qquad k \in K, t \in \mathcal{T} \tag{42}$$

$$\sum_{k \in K} y_{wk}^t \leq 1 \qquad t \in \mathcal{T}, w \in W_u^t \tag{43}$$

$$z_{jk}^t r_{js} \leq \sum_{w \in W_u^t} p_{ws} y_{wk}^t + \sum_{w \in W \backslash W_u^t} e_{wk}^t p_{ws} \qquad j \in J, s = 1, \ldots, S, k \in K, t \in \mathcal{T} \tag{44}$$

$$\sum_{w \in W_u^t} y_{wk}^t + \sum_{w \in W \backslash W_u^t} e_{wk}^t \geq y_j^{min} z_{jk}^t \qquad j \in J, k \in K, t \in \mathcal{T} \tag{45}$$

$$\sum_{q=1}^{Q} z_{jkq}^t = z_{jk}^t \qquad j \in J, k \in K, t \in \mathcal{T} \tag{46}$$

$$\sum_{w \in W_u^t} y_{wk}^t + \sum_{w \in W \backslash W_u^t} e_{wk}^t = \sum_{q=1}^{Q} q z_{jkq}^t \qquad j \in J, k \in K, t \in \mathcal{T} \tag{47}$$

$$v_j^{max} = \sum_{t \in \mathcal{T}} \sum_{k \in K} v_{jk}^t \sum_{q=1}^{Q} \Delta_q \times z_{jkq}^t \qquad j \in J \tag{48}$$

$$v_{jk}^t \geq 0 \qquad j \in J, k \in K, t \in \mathcal{T} \tag{49}$$

$$z_{jk}^t \begin{cases} = 1 \text{ if } j \text{ is assigned to } k \text{ on } t < t'_j \text{ in } s', \\ \leq 1 \text{ if } j \text{ is assigned to } k \text{ on } t = t'_j \text{ in } s', \\ = 0 \text{ otherwise,} \end{cases} \qquad j \in J, k \in K, t \in \mathcal{T} \tag{50}$$

$$y_{wk}^t \in \{0,1\} \qquad k \in K, t \in \mathcal{T}, w \in W_u^t \tag{51}$$

$$z_{jkq}^t \in \{0,1\} \qquad j \in J, k \in K, q = 1, \ldots, Q, t \in \mathcal{T}. \tag{52}$$

Objective function (41) minimizes the sum of two components. The first term measures the total service time needed to serve all jobs, whereas the second term minimizes the sum of the penalties incurred for serving a job across more days than the minimum. Constraints (42) guarantee that for each team $k \in K$ routed on day t, the sum of the actual service times provided is not larger than the duration $\overline{T}$ of the work shift minus the total traveling time of the corresponding route. Constraints (43) impose that on each day t, each unassigned worker $w \in W_u^t$ is associated to at most one team. Constraints (44) establish that for each job j served by team $k \in K$ on day t, the team must be composed of workers with the required skills. Constraints (45) require that for each job j served by team $k \in K$ on day t, the number of workers composing the team must not be smaller than the prescribed lower bound. Constraints (46) impose that for each job j served on day t by team $k \in K$, exactly one variable z_{jkq}^t takes value one. For each job j served on day t by team $k \in K$, constraint (47) forces the binary variable z_{jkq}^t corresponding to the number q of workers assigned to team k to take value 1. At the same time, it limits the number of workers that can be accommodated onto each vehicle to be at most equal to Q. For each job j, constraint (48) relates the required service time with the number of workers assigned to each team serving the job on each day. Finally, constraints (49)–(52) define the nature and domain of the decision variables. In particular, constraints (50) set the value of variables z_{jk}^t according to solution s'. Note that for each job j, constraint $z_{jk}^{t'_j} \leq 1$ allows to not serve job j the last day t'_j, according to solution s', if this is feasible.

Note that the RHS of constraints (48) contains the non-linear term $v_{jk}^t \sum_{q=1}^{Q} \Delta_q z_{jkq}^t$. Such a non-linearity can be transformed into linear form by using standard mathematical programming approaches that require the introduction of the following auxiliary variables and constraints. Let us introduce the (continuous) non-negative variables $u_{jkq}^t \geq 0$, with $j \in J, k \in K, q = 1, \ldots, Q, t = 1, \ldots, T$, with the meaning $u_{jkq}^t = \Delta_q v_{jk}^t z_{jkq}^t$. Then, constraints (48) can be replaced by the following linear inequalities:

$$v_j^{max} = \sum_{t \in \mathcal{T}} \sum_{k \in K} \sum_{q=1}^{Q} u_{jkq}^t \qquad j \in J \tag{53}$$

$$0 \leq u_{jkq}^t \leq M z_{jkq}^t \qquad j \in J, k \in K, q = 1, \ldots, Q, t \in \mathcal{T} \tag{54}$$

$$u_{jkq}^t \leq \Delta_q v_{jk}^t \qquad j \in J, k \in K, q = 1, \ldots, Q, t \in \mathcal{T} \tag{55}$$

$$\Delta_q v_{jk}^t \leq u_{jkq}^t + M(1 - z_{jkq}^t) \quad j \in J, k \in K, q = 1, \ldots, Q, t \in \mathcal{T}. \tag{56}$$

If on day t team $k \in K$ serving job $j \in J$ is not composed of q workers (i.e., $z_{jkq}^t = 0$), then variable u_{jkq}^t is forced to take value zero by inequality (54). On the other side, if there exists a q such that $z_{jkq}^t = 1$, variable u_{jkq}^t takes value $\Delta_q v_{jk}^t$ forced by constraints (55) and (56). Note that in constraints (54) one can set $M = \min\{v_j^{max}, \Delta_q \times \overline{T}\}$, whereas in constraints (56) one can set $M = \Delta_Q \min\{v_j^{max}, \times \overline{T}\}$.

An optimal solution to problem (41)–(52), with constraints (48) replaced by (53)–(56), indicates the new team compositions. The partly destroyed solution s' is then repaired by modifying the team compositions accordingly.

3.4 Selecting a destroy-repair operator pair

As mentioned above, and in contrast with a standard implementation of an ALNS algorithm, in our implementation scores and weights are assigned to pairs of destroy and repair operators. Particularly, we consider every pair composed of a destroy job operator {Random_Job_Removal, Worst_Job_Removal, Related_Jobs_Removal} and a repair job operator {Greedy_Job_Insertion, Regret_Job_Insertion-$\kappa = 2$, Regret_Job_Insertion-$\kappa = 3$, Regret_Job_Insertion-$\kappa = 4$, Regret_Job_Insertion-$\kappa = 5$}, where each of the latter is duplicated to contemplate the two variants (the deterministic one and the one with a random noise). Finally, we consider every pair comprising a destroy team operator {Random_Team_Removal, All_Teams_Removal} and a repair team operator {Team_Repair}.

Let $\mathcal{N}_{d,r}$ be the number of destroy-repair operator pairs considered. In each iteration of the ALNS, a roulette-wheel selection routine is invoked to select the pair (d, r) to employ. A weight ω_{dr}, with $dr = 1, \ldots, \mathcal{N}_{d,r}$, is associated with each destroy-repair operator pair (d, r). Thus, in each iteration a given pair (d, r) is selected with probability:

$$\frac{\omega_{dr}}{\sum_{dr'=1}^{\mathcal{N}_{d,r}} \omega_{dr'}}. \tag{57}$$

As in Røpke and Pisinger (2006), weights ω_{dr} are dynamically adjusted during the search. Particularly, in addition to the weights, a score ψ_{dr} is associated with each pair (d, r). At the beginning, weights ω_{dr} are all set to one, whereas scores ψ_{dr} are all set to zero. At the end of each iteration, score $\psi_{dr'}$ associated with the pair $(d, r)'$ employed is updated as follows. If the solution s^{new} found after applying this pair improved upon the best-known solution s^*, then score $\psi_{dr'}$ is incremented by σ_1. Conversely, if s^{new} did not improve upon s^*, but the solution is accepted in a simulated annealing fashion, the score is updated as follows. If the value of s^{new} is better than that of the current solution s, then score $\psi_{dr'}$ is incremented by σ_2, otherwise the score is increased by σ_3. The score is not modified if solution s^{new} is not accepted. At every $iter_{upd}$ iterations, the weight ω_{dr} of each pair (d, r) is adjusted according to the formula $\omega_{dr} = (1 - p_{react}) \times \omega_{dr} + p_{react} \times \frac{\psi_{dr}}{\max\{1, it_{empl}\}}$, where ψ_{dr} is the score that pair (d, r) achieved in the last $iter_{upd}$ iterations, it_{empl} is the number of times this pair has been employed out of the last $iter_{upd}$ iterations, and p_{react} is the reaction factor (as called by Røpke and Pisinger, 2006). After adjusting all weights, scores ψ_{dr} are all re-initialized to zero.

4 Experimental analysis

This section is devoted to the presentation and discussion of the computational experiments. They were conducted on a Workstation HP Intel(R)-Xeon(R) at 3.5GHz with 64 GB RAM (Win 10 Pro, 64 bits). The processor is equipped with 6 physical cores. Only one thread was used in all the experiments. The MPWSRP model has been implemented in C++ and by using the ILOG Concert Technology API. The ALNS heuristic has been implemented in C++ and the optimization models used in the repair operators have been implemented using CPLEX Concert Technology API 12.6.0.0 with default parameter values.

This section is organized as follows. In Section 4.1, the set of instances used in the computational experiments is described. Section 4.2 reports the computational results conducted on small- and large-size instances. Finally, the results of our case study are reported in Section 4.3.

4.1 Testing environment

Since we introduce the MPWSRP, no benchmark instances are available from the literature. Hence, we have generated two sets of instances from those originally proposed by Kovacs et al. (2012). In particular, we have generated one set of small-size instances, which are solved with ALNS and CPLEX, and one set of large-size instances, which are solved only with ALNS. Finally, we have conducted some experiments using five data sets provided by the company described in the Introduction.

The small- and large-size instances have been generated as follows. Regarding the jobs data, Kovacs et al. (2012) have proposed 36 instances that are suitable for the MPWSRP (we have considered only the "team" version). These instances have been generated starting from 12 of the Solomon's instances for the VRP with time windows. Then, for each of these instances, they have generated three instances for their problem characterized by different combinations of skill domains and proficiency levels, namely 5 skills (and 4 proficiency levels), 6 skills (and 6 levels), and 7 skills (and 4 levels). Each of these instances comprises one depot and 100 jobs. From these instances, we have kept the coordinates of each vertex and adapted the skill requirements of each job as follows. In the original instances, several skill domains are available and, for each skill domain, different proficiency levels are defined. For each combination of skill domain and proficiency level, these instances indicate the required number of workers. Recall that in the MPWSRP we only need to define the 0-1 parameter r_{js}. To this aim, for each job $j \in J$ and skill domain $s \in S$, if in the original instance the job requires at least one worker with that skill (considering the sum over all proficiency levels), then in the corresponding instance for the MPWSRP job j requires skill s (i.e., $r_{js} = 1$). Otherwise, that parameter is set to zero (i.e., $r_{js} = 0$). Travel costs and times are equal to Euclidean distances, truncated after the second decimal digit.

We set the length $\overline{T}$ of the daily work shift equal to 480 minutes. To allow the presence of jobs whose service spans across multiple days, the service time v_j^{max} of each job j has been generated as follows. Firstly, a random number (equally distributed from 0 to 1) is generated. When this number was smaller than 0.75, then a second random number equally distributed from 90 to 500 was extracted. The latter number is the service time (in minutes) of job j. On the other side, when the first random number was larger or equal than 0.75, the service time of job j was generated randomly in the interval [500, 900]. Furthermore, for each job j, the minimum number of workers required to perform the job (parameter y_j^{min}) is an integer number randomly generated between 1 and 2 workers, whereas its priority p_j is an integer number randomly generated from set $\{1, 2, 3, 4\}$. Along the lines described above, we have generated 36 data sets detailing different job configurations.

As far as the workers data are considered, we have proceeded as follows. Also here, we have started from those proposed by Kovacs et al. (2012). Considering the three instances they proposed, if a worker w had a proficiency level for a given skill domain s larger than half of the maximum possible level (e.g., if the maximum proficiency level for skill s is 5, then larger than 2.5), then in the corresponding instance of the MPWSRP worker w is proficient in skill domain s (i.e., $p_{ws} = 1$). Otherwise, that parameter is set to zero (i.e., $p_{ws} = 0$). Regarding the efficiency, in terms of the reduction of the service times, of a team composed of $|k|$ workers (parameter Δ_k), it is computed as a simple linear function of the number of workers composing the team: $\Delta_k = 1 + 0.5 * (|k| - 1)$. Along these lines, we have generated three data sets detailing three different worker configurations.

The goal of the computational experiments on the small-size instances is threefold: calibrating the control parameters of ALNS, validating its performance against CPLEX, and determining the largest-size instances that can be solved with CPLEX in reasonable computing times. We have then created 108 instances by combining different jobs, skills, and workers configurations. More specifically, these instances are equally distributed among data sets comprising 8, 10, 12 and 15

jobs and one depot. Each of the latter sub-groups is made up of 27 instances created as follows. First, we randomly select one job configuration from the 36 data sets generated. For the selected job configuration, we have considered all the three related configurations corresponding to the different sets of skills (i.e., with 5, 6, and 7 skills), and selected the first n_s jobs, with $n_s = 8, 10, 12, 15$. Each of the three job configurations is then coupled with three worker data sets, all having the same set of skills but comprising a different number of workers. Particularly, the three worker data sets have been created selecting the first 5, 6, and 10 workers, respectively, listed in the data sets generated as detailed above. At this point, 36 instances have been created characterized by different numbers of jobs, skills, and workers. This process is repeated three times to create 108 instances. Each of the small-size instances has been solved by setting the maximum number Q of workers that can be accommodated onto a vehicle equal to 3. As the number of variables of the MPWSRP model (1)-(24) grows very rapidly with the number of days composing the planning horizon, we have solved each instance increasing iteratively the number of days (parameter T) and the number of vehicles available (parameter D until 4) while CPLEX was able to find a feasible solution within a time limit. All small-size instances have then been solved by means of CPLEX with a time limit of 3,600 seconds.

The large-size instances have been generated similarly. Based on the results obtained on the small-size instances, where we did not detect any sizable impact of the number S of skills on the computing times required both by CPLEX and ALNS (more details are provided in Section 4.2), for the large-size instances we have considered only the job and worker configurations with the same number of skills. In more details, we have selected the 12 job configurations, out of the 36 data sets generated as described above, having a number of skills S equal to 5. From each of these 12 job configurations, we have generated 3 data sets: the first obtained by selecting the first 25 jobs, the second by selecting the first 50 jobs, and the third that considers all 100 jobs. Each of the latter instances is then coupled with three worker data sets, all having a number of skills $S = 5$ but comprising a number of workers equal to 10, 15, and 20, respectively. In all large-size instances the maximum number Q of workers that can be accommodated onto each vehicle is equal to 3, whereas the number D of vehicles available is set according to the number $|W|$ of workers: $D = 4$ when $|W| = 10$, $D = 5$ when $|W| = 15$, and $D = 6$ when $|W| = 20$. The number T of days composing the planning horizon is set equal to 10 for all instances with 25 jobs, equal to 15 for all instances with 50 jobs, and equal to 20 for all instances with 100 jobs with the only exception of instance C101 with 10 workers (in this case, the small number of workers available made, probably, impossible for ALNS to find any feasible solution with T smaller than 25 days). Altogether, we have generated and solved 108 large-size instances.

The data sets used for the case study are detailed below in Section 4.3. All job and worker configurations in our test bed are publicly available at the web page `http://or-brescia.unibs.it/instances`.

4.2 Computational results

This section is devoted to the illustration and comment of the computational results. It is divided into two parts, concerning the experiments conducted on the small- and large-size instances.

As mentioned above, the control parameters of ALNS have been calibrated solving the small-size instances. After extensive preliminary experiments, we have found that the best performing configuration is the one reported in Table 1.

The results of the experiments conducted on the small-size instances are summarized in Table 2. For each group of 9 instances, clustered based on the number of jobs (n) and the number of workers ($|W|$), Table 2 compares the average performance of CPLEX and ALNS. The detailed computational

Param.	Description	Value	Param.	Description	Value
	Initial solution - Phase One			*Repair operators*	
π_α	Weight of criticality	0.1	$\hat{\pi}_\alpha$	See *Initial solution* - Phase Two	
π_β	Weight of difficulty	1	$\hat{\pi}_\delta$	See *Initial solution* - Phase Two	
π_γ	Weight of similarity	1	$\hat{\pi}_\rho$	See *Initial solution* - Phase Two	
π_δ	Weight of distance	1	η	Amount of noise in objective	0.025
θ	Safety level of workers	0.2		function randomization	
	Initial solution - Phase Two		$h_j, j \in J$	Penalty for serving jobs	70
$\hat{\pi}_\alpha$	Weight of criticality	5		across too many days	
$\hat{\pi}_\beta$	Weight of difficulty	1		*General ALNS*	
$\hat{\pi}_\delta$	Weight of insertion cost	10	τ_{start}	Initial temperature	$1.05 \times z(s)^\dagger$
$\hat{\pi}_\phi$	Weight of job splitting	10	c	Cooling rate	0.9996
$\hat{\pi}_\rho$	Weight of priority delay	5	p_{react}	Reaction factor	0.05
	Destroy operators - Worst_Job_Removal		σ_1	Update score: New best-known	4
p_{worst}	Degree of randomization	3	σ_2	Update score: New accepted is	1
	Destroy operators - Related_Jobs_Removal			better than the current	
$\bar{\pi}_d$	Weight of traveling cost	0.7	σ_3	Update score: New accepted is	0.1
$\bar{\pi}_t$	Weight of traveling time	0.7		worse than the current	
$\bar{\pi}_\gamma$	Weight of skills required	0.5	$iter_{upd}$	Number of iterations in	100
$\bar{\pi}_p$	Weight of priority	2.5		each segment	
$\bar{\pi}_w$	Weight of number of workers	0.3	$iter_{max}$	Maximum number of iterations	5000
$\bar{\pi}_a$	Weight of starting day	2.5	$iter_{maxNoImpr}$	Maximum consecutive iterations	1000
p_{rel}	Degree of randomization	6		without improvement	

$\dagger : z(s)$ is the total distance traveled in the initial solution.

Table 1: ALNS: Best parameters configuration.

results are reported in Appendix A. In Table 2, two statistics are reported for measuring CPLEX performance: *# Inf.*, which counts the number of instances where the method did not find any feasible solution within the time limit; and *CPU (secs.)*, which measures the average computing time (in seconds). As far as the ALNS performance is concerned, 5 further statistics are provided:

- *Gap %*: measures the average percentage gap, computed for a single instance as $100 \times \frac{z(s_A^*)-z(s_C^*)}{z(s_C^*)}$, between the value of the best solution produced by ALNS ($z(s_A^*)$) and the value of the best solution found by CPLEX ($z(s_C^*)$);
- *# Impr.*: counts the number of instances where the solution determined by ALNS is better than the one produced by CPLEX (i.e., "Gap %" is negative);
- *Best Impr.*: is the best improvement achieved in an instance by ALNS over CPLEX;
- *# Worsts*: counts the number of instances where the solution determined by ALNS is worse than the one produced by CPLEX (i.e., "Gap %" is positive);
- *Worst Gap*: is the worst error produced in an instance by ALNS compared to CPLEX.

A general conclusion that can be drawn from the results shown in Table 2 is that ALNS finds solutions that, on average, are of similar quality compared to those produced by CPLEX. Statistic "Gap %" is always smaller than 1% (error), with the only exception of the 3 instances solved by CPLEX with $n = 15$ and $|W| = 5$ (note that on this group, CPLEX did not find any feasible solution for 6 out of the 9 instances). Additionally, statistic "Gap %" is sometimes smaller than -1% (improvement), see, for instance, the instances with $n = 12$ and $|W|$ equal to 6 and 10. Note also that the average value of statistic "Gap %" computed over all the small-size instances is equal to -0.24% (see Table 7), highlighting a small average improvement of ALNS over CPLEX when all the instances

Instance data			CPLEX		ALNS						
n	|W|	# Inst.	# Inf.	CPU (secs.)	# Inf.	Gap %	# Impr.	Best Impr.	# Worsts	Worst Gap	CPU (secs.)
8	5	9	0	2,224.9	0	0.10%	0	0.00%	2	0.86%	238.7
	6	9	0	1,342.4	0	0.90%	0	0.00%	5	2.73%	212.7
	10	9	0	3,600.0	0	-0.02%	1	-2.29%	1	2.12%	246.6
Average (Total)		(27)	(0)	2,389.1	(0)	0.33%	(1)		(8)		232.7
10	5	9	0	3,600.0	0	-0.30%	2	-5.01%	4	1.31%	277.2
	6	9	0	2,730.0	0	0.29%	1	-4.00%	2	3.76%	255.8
	10	9	0	3,600.0	0	-0.12%	4	-4.00%	3	1.92%	327.4
Average (Total)		(27)	(0)	3,310.0	(0)	-0.04%	(7)		(9)		286.8
12	5	9	1	3,600.0	0	-0.45%	4	-4.39%	4	2.46%	320.3
	6	9	2	3,600.1	0	-1.06%	3	-8.89%	2	4.24%	326.7
	10	9	4	3,600.1	0	-3.90%	4	-8.14%	1	0.97%	393.4
Average (Total)		(27)	(7)	3,600.1	(0)	-1.53%	(11)		(7)		346.8
15	5	9	6	3,600.0	0	2.01%	0	0.73%	3	3.31%	464.8
	6	9	2	3,600.0	0	-0.46%	5	-3.57%	2	1.80%	450.8
	10	9	9	3,600.1	0	–	–	–	–	–	520.2
Average (Total)		(27)	(17)	3,600.0	(0)	0.28%	(5)		(5)		478.6

Table 2: Average results for the small-size instances.

in the test bed are considered. More importantly, ALNS finds such solutions in considerably shorter computing times: the average computing time required by CPLEX is equal 3,224.8 seconds, whereas it is equal to 336.2 seconds for ALNS, indicating that the latter can produce, on average, solutions of similar quality than the former in computing times that are one order of magnitude faster. Moreover, our heuristic provides solutions to all instances, which is not the case for CPLEX, that fails to find any feasible solution for larger instances. This skews some of the statistics of the table, but should be considered when judging the quality of the ALNS algorithm.

Instance data			CPLEX		ALNS	
n	S	# Inst.	# Inf.	CPU (secs.)	Gap %	CPU (secs.)
8	5	9	0	2,147.3	0.71%	208.0
	6	9	0	2,635.6	-0.24%	233.1
	7	9	0	2,384.5	0.51%	256.9
10	5	9	0	3,089.3	-0.81%	290.7
	6	9	0	3,600.0	0.32%	282.4
	7	9	0	3,240.8	0.36%	287.3
12	5	9	3	3,600.1	-4.43%	396.4
	6	9	3	3,600.1	-0.89%	331.1
	7	9	1	3,600.0	0.17%	313.0
15	5	9	6	3,600.0	1.94%	445.0
	6	9	8	3,600.0	0.22%	474.8
	7	9	3	3,600.1	-1.01%	510.0

Table 3: Small-size instances: Impact of the number of skills.

Looking in more details into the figures shown in Table 2 one can notice that, as expected, computing times required by CPLEX increase with the number n of jobs, and also tend to increase, for a given number of jobs, with the number $|W|$ of workers. The latter is also an expected outcome since increasing the number of workers increases the number of teams created and, as consequence,

the number of binary variables x^t_{ijk} and z^t_{jk}. Finally, it is worth highlighting that CPLEX does not find any feasible solution within the time limit for some of the instances with 12 jobs or more, and cannot find any feasible solution for those with 15 jobs and 10 workers. As far as ALNS is concerned, also the computing times for the proposed matheuristic increase, on average, with the number n of jobs, but they remain very short. Indeed, ALNS achieves the largest average computing time for the group of instances with 15 nodes and 10 workers, which is smaller than 9 minutes. In terms of solution quality, the average value of statistic "Gap %" (as well as statistics "# Impr." and "Best Impr.") tends to improve with the number of jobs, ranging from 0.33% ($n = 8$) to -1.53% ($n = 12$). Note that we did not consider the group with $n = 15$ since, for the latter, CPLEX did not find any feasible solution within the time limit for 17 out of the 27 instances.

Table 3 summarizes the computational results on the small-size instances based on the number n of jobs and, then, on the number S of skills, trying to identify any impact of the number of skills on the performance of CPLEX and ALNS. Nevertheless, the results indicate that there is no clear influence of the number of skills on the performance of both CPLEX and ALNS. Consequently, we have conducted the experiments on the large-size instances discussed below considering only a single number of skills, namely $S = 5$.

Finally, we have conducted also experiments on the small-size instances to assess the impact of the individual operators in our ALNS algorithm. Table 4 shows the average increase, in percentage, of the best solution value when individual operators are eliminated from ALNS. According to these results, the elimination of the operators that modify the team compositions have the highest impact on the solution quality (3.10%). Note that eliminating these operators ALNS is forced to use the teams constructed in the initial solution, and the greedy heuristic Construct_Team to create teams to assign to unused vehicles. The second operator with highest impact is the Related_Jobs_Removal operator (1.41%), followed by the Worst_Job_Removal operator (1.13%). All the remaining operators show an impact smaller than 1%. Among the insertion operators, the Regret_Job_Insertion with $\kappa = 2$ and without the random noise has the smallest impact (0.37%, very similar to the impact of the Greedy_Job_Insertion operator), whereas the use of its randomized variant has the largest deterioration (0.87%).

Operator eliminated	**Avg. Increase**	**Operator eliminated**	**Avg. Increase**
Random_Job_Removal	0.70%	Worst_Job_Removal	1.13%
Related_Jobs_Removal	1.41%	Random_Team_Removal, All_Teams_Removal, Team_Repair	3.10%
Greedy_Job_Insertion	0.38%	Regret_Job_Insertion (with $\kappa = 2$ and no noise)	0.37%
Regret_Job_Insertion (with $\kappa = 2$ and noise)	0.87%	Regret_Job_Insertion (with $\kappa = 3$ and no noise)	0.82%
Regret_Job_Insertion (with $\kappa = 3$ and noise)	0.53%	Regret_Job_Insertion (with $\kappa = 4$ and no noise)	0.72%
Regret_Job_Insertion (with $\kappa = 4$ and noise)	0.53%	Regret_Job_Insertion (with $\kappa = 5$ and no noise)	0.63%
Regret_Job_Insertion (with $\kappa = 5$ and noise)	0.55%		

Table 4: Small-size instances: Average increase of the best solution value when individual operators are eliminated from ALNS.

We now turn our attention to the large-size instances. As mentioned above, the latter have been solved only by means of ALNS, as CPLEX cannot address instances of these sizes. The detailed computational results for the large-size instances are reported in Table 5. This table is divided into 4 main parts. The first 5 columns provide information on the instance, which are common for the three instances with 25, 50, and 100 jobs. The following 3 groups of 4 columns refer to the instances with 25, 50, and 100 jobs, respectively. It is, obviously, impossible to benchmark the quality of the solutions found, as no solutions for comparison are available. Regarding computing times, besides increasing with the number of jobs, they are reasonable given the size of the instances solved. Indeed, ALNS solves, on average, the largest-size instances, that comprise 100 jobs and 20 days, in less than 2,559 seconds, and, in the worst-case, in less than 5,157 seconds.

Inst. Details					ALNS n=25, T=10		ALNS n=50, T=15		ALNS n=100, T=20	
Name	**S**	**|W|**	**Q**	**D**	$\mathbf{z(s^*_A)}$	**CPU (secs.)**	$\mathbf{z(s^*_A)}$	**CPU (secs.)**	$\mathbf{z(s^*_A)}$	**CPU (secs.)**
C101	5	10	3	4	1,156.60	142.7	2,395.93	192.2	$7,119.36^{\dagger}$	723.1
C101	5	15	3	5	1,173.10	49.0	2,370.95	70.8	6,605.81	147.0
C101	5	20	3	6	1,149.96	38.8	2,324.66	132.4	6,315.21	229.8
C103	5	10	3	4	1,012.82	375.0	1,784.51	257.8	5,372.88	2,219.3
C103	5	15	3	5	1,166.29	493.3	1,779.31	1,224.1	4,735.18	3,714.4
C103	5	20	3	6	926.51	641.3	1,679.92	39.0	4,555.07	4,152.6
C201	5	10	3	4	1,099.05	401.5	2,245.94	354.3	5,678.64	3,003.0
C201	5	15	3	5	915.79	422.6	1,951.52	32.6	4,867.07	275.6
C201	5	20	3	6	860.12	41.4	1,964.75	39.7	4,735.34	100.5
C203	5	10	3	4	1,133.17	374.9	2,377.55	958.9	5,376.93	2,762.1
C203	5	15	3	5	957.08	18.2	1,962.82	33.2	4,802.62	3,361.6
C203	5	20	3	6	1,014.26	93.0	2,395.38	1,602.2	4,785.29	4,340.3
R101	5	10	3	4	925.14	12.4	2,541.97	1,193.5	4,692.32	2,927.2
R101	5	15	3	5	882.32	18.9	2,444.42	1,460.3	3,929.64	69.2
R101	5	20	3	6	904.83	16.3	2,270.76	129.8	4,157.28	4,375.7
R103	5	10	3	4	1,219.10	377.2	2,475.45	1,139.0	4,618.42	2,866.2
R103	5	15	3	5	1,034.12	45.4	2,335.67	1,518.8	4,125.15	3,608.2
R103	5	20	3	6	1,048.30	21.1	2,383.41	1,566.0	42,60.01	963.3
R201	5	10	3	4	1,224.41	406.8	2,516.52	1,085.6	4,402.58	2,872.3
R201	5	15	3	5	1,099.84	559.3	2,095.79	1,321.6	3,824.04	3,228.1
R201	5	20	3	6	997.53	44.5	2,209.80	1,768.0	3,826.29	4,175.1
R203	5	10	3	4	1,224.82	392.4	2,566.95	1,092.0	4,513.19	2,734.5
R203	5	15	3	5	1,116.47	458.0	2,318.21	1,180.7	3,787.93	3,349.1
R203	5	20	3	6	887.82	27.2	2,574.26	1,737.5	3,732.15	117.6
RC101	5	10	3	4	2,008.10	406.7	4,054.09	1,181.7	6,461.89	3,010.6
RC101	5	15	3	5	1,357.23	59.2	3,631.07	1,375.1	5,937.49	3,517.2
RC101	5	20	3	6	1,414.02	192.5	3,201.09	38.8	5,813.06	4,810.1
RC103	5	10	3	4	1,454.15	19.1	4,086.92	1,080.0	6,426.71	2,973.9
RC103	5	15	3	5	1,539.90	502.6	3,620.08	123.0	5,689.23	3,672.8
RC103	5	20	3	6	1,318.39	42.2	3,419.74	1,617.8	5,263.81	624.5
RC201	5	10	3	4	1,561.56	399.7	3,586.26	1,453.4	6,561.05	3,374.4
RC201	5	15	3	5	1,262.38	24.2	3,413.55	1,322.6	5,945.26	3,155.9
RC201	5	20	3	6	1,518.69	650.4	3,250.04	42.0	6,035.67	5,156.9
RC203	5	10	3	4	1,428.05	344.2	3,572.80	1,166.9	5,805.73	2,797.5
RC203	5	15	3	5	1,374.46	401.9	3,214.45	1,193.7	5,261.52	3,275.4
RC203	5	20	3	6	1,163.81	16.6	3,241.28	1,499.9	4,888.48	139.8
Average CPU (secs.)						237.0		895.1		2,558.4

$\dagger : T = 25$.

Table 5: Large-size instances: Detailed computational results.

4.3 Case study

This section is devoted to the presentation and discussion of a case study conducted on real-world data. As mentioned in the introduction, the target company operates in the province of Québec, Canada. The company installs, maintains, and repairs electrical infrastructures. Given the range of services provided, each job request received by the company requires one or more skills, identified among a set of 4 different ones. Once a job request is received, a priority code is assigned ranging from 1 (the most urgent, typically the repair of critical infrastructure) to 4 (the less urgent, typically jobs that can be postponed without interrupting the operation of other activities). Then, the service time needed to completely serve the job is estimated. The company imposes that each job must be performed by at least two workers, and is willing to assign more workers, if possible, to speed up service times. As of the time of this analysis, the company has hired 17 skilled workers, each proficient in one or more of the 4 aforementioned skills.

At the beginning of each planning horizon (typically between two to three weeks), the scheduler at the company has to decide when each job request will be served. Simultaneously, the scheduler, on each day, has to group workers into teams, to assign each team to a set of jobs whose service is scheduled on that day, and to determine a routing plan for each team. The company allows that different teams work on the same job on different days. Each team leaves the company's depot around 8 A.M., and must respect a maximum work shift of 8 hours (which includes also the time to travel to job locations). Teams travel on the homogenous vehicles owned by the company. The latter communicated us that, to reach job locations, the average travel speed of the vehicles is 60 kilometers per hour.

We organized the experiments on the case study as follows. The company has provided us with the jobs data spanning over a period of several weeks. We have divided these data into 5 job configurations comprising 60 jobs each. These jobs are rather widespread around Québec City. As an example, Figure 3 shows the location of each job in instance CaseStudy1. We have computed the great-circle distance between any pair of vertices by means of the haversine formula. According to the company's attitude, the objective is to minimize the total distance driven by the fleet of vehicles available. Each of the 5 job configurations is coupled with the only worker configuration available (including all the 17 workers), and then solved with ALNS assuming a planning horizon T of 15 days and considering that the maximum number Q of workers that can be accommodated onto each vehicle is 3.

The foremost outcomes of these experiments are shown in Table 6. Particularly, the column headings have the following meaning. Column $z(s_A^*)$ shows the value of the best solution found by ALNS (that is, the total distance, in kilometers, driven by the vehicles), whereas column *CPU (secs.)* indicates the computing time (in seconds). The next columns provide some statistics regarding the solution obtained. More specifically, the number of days required to serve all the jobs is reported in column *# days used*; the average number of jobs served on each of the latter days is shown in column *Av. # jobs per day*; the average number of teams used on each day is indicated in column *Av. # teams per day*; the average number of workers assigned on each day is reported in column *Av. # work. per day*; and, finally, the average daily work shift length, in minutes, is shown in column *Av. work shift (min.)*.

The results reported in Table 6 show that a number of days between 11 and 13, depending on the data set, are necessary to completely serve all jobs. The number of jobs served per day is relatively small (ranging from 8.15 to 10.00). This can be motivated by the moderately long service and driving times that make it infeasible to serve a larger number of jobs per day. This claim is supported by the value of the average daily work shift length, which is always very close to its maximum limit of 480

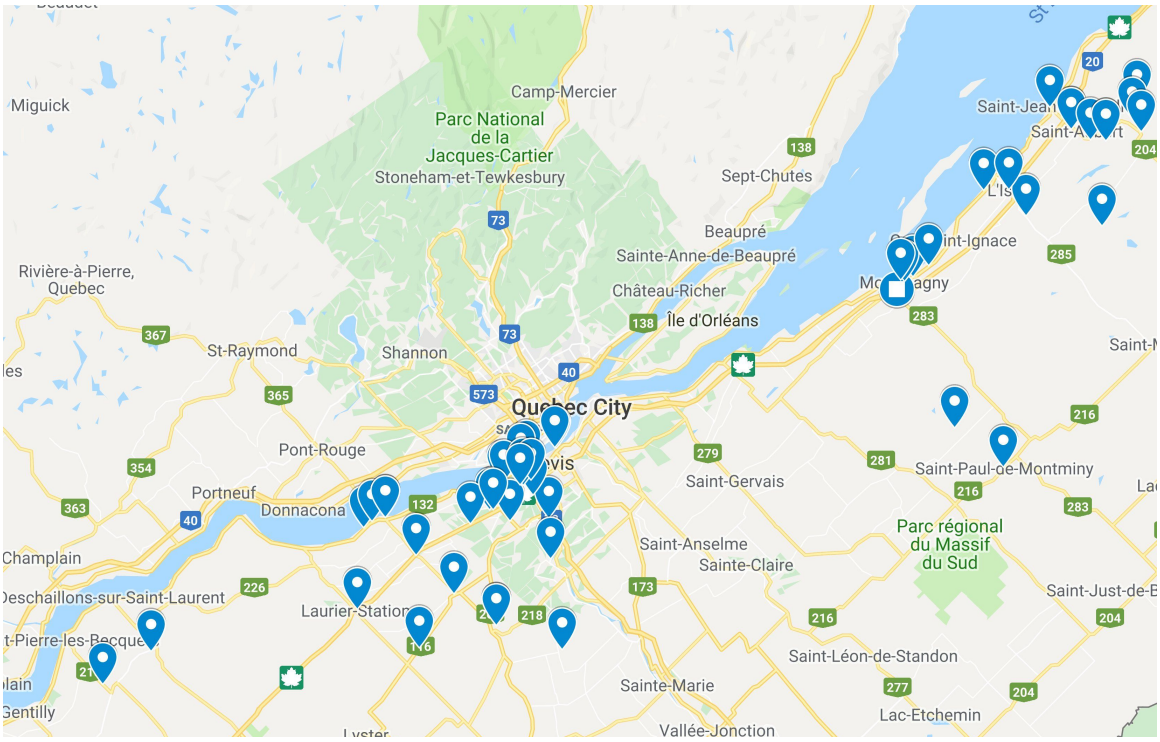

Figure 3: Map of the job locations in instance CaseStudy1. *Source:* Google Maps.

minutes. The remaining statistics indicate an intensive usage of the resources available. The average number of teams used daily ranges from 4.75 to 5.46 (recall that 6 is the daily maximum value), the average number of workers assigned daily varies from 13.67 to 15.55 (out of the 17 available), and the average daily work shift length ranges from 428.19 to 461.91 seconds.

	Inst. Details							CPU	# days	Av. # jobs	Av. # teams	Av. # work.	Av. work shift
Name	S	|W|	Q	D	n	T	$z(s^*_A)$	(secs.)	used	per day	per day	per day	(min.)
CaseStudy1	4	17	3	6	60	15	9,470.83	51.3	11	10.00	5.45	15.55	461.91
CaseStudy2	4	17	3	6	60	15	10,547.82	45.7	13	8.15	5.46	15.54	428.19
CaseStudy3	4	17	3	6	60	15	7,847.10	44.8	12	8.25	4.75	13.67	450.18
CaseStudy4	4	17	3	6	60	15	8,216.81	43.7	11	8.73	5.09	14.55	459.50
CaseStudy5	4	17	3	6	60	15	8,606.29	49.2	12	8.25	4.92	14.08	455.83

Table 6: Case study: Computational results.

5 Conclusions

Inspired by a real-world application faced by a service company installing and maintaining electrical infrastructures, we have introduced a new variant in the class of routing and scheduling problems of skilled workforce crews. We have named this variant the Multi-Period Workforce Scheduling and Routing Problem (MPWSRP). The main original characteristics of MPWSRP are the multi-period setting, the presence of jobs whose service spans across multiple time periods, and service times that are a function of team composition. We have proposed a Mixed-Integer Linear Programming (MILP) formulation for the MPWSRP, and have developed a matheuristic based on ALNS in which some of its operators are based on the solution to optimality of some easy-to-solve mathematical models.

Extensive computational experiments have been carried out to validate the performance of ALNS. The results have shown that an off-the-shelf solver can solve in reasonable computing times only small-size instances of the above-mentioned MILP. ALNS is competitive with the solver on this set of instances, producing solutions of similar average quality in considerably shorter computing times. We have also conducted some further computational experiments by solving large-size instances with ALNS. Finally, we have analyzed a case study based on the real-world data provided by the target company.

Acknowledgments

We wish to thank our contact persons at the company partner for providing us with the data necessary to conduct the case study. This work was partly funded by the Canadian Natural Sciences and Engineering Research Council under grants 2015-04893 and 2019-00094.

References

R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

T. Bektaş, G. Laporte, and D. Vigo. Integrated vehicle routing problems. *Computers & Operations Research*, 55(0):126, 2015.

P. De Bruecker, J. Van den Bergh, J. Beliën, and E. Demeulemeester. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16, 2015.

J. O. Brunner and J. F. Bard. Flexible weekly tour scheduling for postal service workers using a branch and price. *Journal of Scheduling*, 16(1):129–149, 2013.

R. Bürgy, H. Michon-Lacaze, and G. Desaulniers. Employee scheduling with short demand perturbations and extensible shifts. *Omega*, 89:177–192, 2019.

F. Castaño and N. Velasco. Exact and heuristic approaches for the automated design of medical trainees rotation schedules. *Omega*, forthcoming, 2019.

J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu. Workforce scheduling and routing problems: Literature survey and computational study. *Annals of Operations Research*, 239(1):39–67, 2016.

J.-F. Cordeau, G. Laporte, F. Pasin, and S. Røpke. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409, 2010.

C. E. Cortés, M. Gendreau, L.-M. Rousseau, S. Souyris, and A. Weintraub. Branch-and-price and constraint programming for solving a real-life technician dispatching problem. *European Journal of Operational Research*, 238(1):300–312, 2014.

J.-F. Côté, G. Guastaroba, and M.G. Speranza. The value of integrating loading and routing. *European Journal of Operational Research*, 257(1):89–105, 2017.

A. Dohn, E. Kolind, and J. Clausen. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4): 1145–1157, 2009.

A. Goel and F. Meisel. Workforce routing and scheduling for electricity network maintenance with downtime minimization. *European Journal of Operational Research*, 231(1):210–228, 2013.

B.-I. Kim, J. Koo, and J. Park. The combined manpower-vehicle routing problem for multi-staged services. *Expert Systems with Applications*, 37(12):8424–8431, 2010.

A.A. Kovacs, S.N. Parragh, K.F. Doerner, and R.F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, 15(5):579–600, 2012.

B. Legros, O. Jouini, and Y. Dallery. A flexible architecture for call centers with skill-based routing. *International Journal of Production Economics*, 159:192–207, 2015.

Y. Li, A. Lim, and B. Rodrigues. Manpower allocation with time windows and job-teaming constraints. *Naval Research Logistics*, 52(4):302–311, 2005.

V. Pillac, C. Gueret, and A. L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535, 2013.

M. I. Restrepo, L.-M. Rousseau, and J. Vallée. Home healthcare integrated staffing and scheduling. *Omega*, forthcoming, 2019.

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

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Lecture Notes in Computer Science: Vol. 1520 - 4th International Conference on Principles and Practice of Constraint Programming - CP98*, pages 417–431. Springer, 1998.

H. Tang, E. Miller-Hooks, and R. Tomastik. Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, 43(5):591–609, 2007.

J. Xu and S. Y. Chiu. Effective heuristic procedures for a field technician scheduling problem. *Journal of Heuristics*, 7(5):495–509, 2001.

Appendices

A Small-size instances: Detailed computational results

Instance Details							CPLEX		ALNS		
Name	n	S	|W|	Q	D	T	$z(s^*_C)$	CPU (secs.)	$z(s^*_A)$	Gap %	CPU (secs.)
R101	8	5	5	3	2	3	249.26	297.2	249.26	0.00%	198.4
R101	8	5	6	3	2	3	249.26	19.8	255.73	2.60%	197.2
R101	8	5	10	3	4	2	234.61	3600.0	239.59	2.12%	148.2
R101	8	6	5	3	2	4	258.14	885.0	258.14	0.00%	350.8
R101	8	6	6	3	2	4	258.14	43.8	258.56	0.16%	295.4
R101	8	6	10	3	4	3	255.58	3600.0	255.58	0.00%	326.9
R101	8	7	5	3	2	3	284.51	114.1	284.51	0.00%	189.3

Continued on next page

Instance Details							CPLEX		ALNS		
Name	**n**	**S**	**|W|**	**Q**	**D**	**T**	$\mathbf{z(s^*_C)}$	**CPU (secs.)**	$\mathbf{z(s^*_A)}$	**Gap %**	**CPU (secs.)**
R101	8	7	6	3	2	3	276.26	85.8	283.8	2.73%	184.6
R101	8	7	10	3	4	4	279.72	3600.0	279.72	0.00%	218.4
C103	8	5	5	3	2	4	154.38	3600.0	154.38	0.00%	342.8
C103	8	5	6	3	2	3	155.46	2699.8	158.09	1.69%	232.7
C103	8	5	10	3	4	2	151.77	3600.0	151.77	0.00%	261.2
C103	8	6	5	3	2	3	151.44	3600.0	151.44	0.00%	154.4
C103	8	6	6	3	2	6	151.44	3600.0	151.44	0.00%	201.3
C103	8	6	10	3	4	2	151.3	3600.0	147.83	-2.29%	159.4
C103	8	7	5	3	2	4	156.97	2601.0	158.32	0.86%	171.4
C103	8	7	6	3	2	3	155.49	659.6	156.97	0.95%	164.8
C103	8	7	10	3	4	2	154.39	3600.0	154.39	0.00%	160.1
RC101	8	5	5	3	2	3	386.06	1727.1	386.06	0.00%	152.2
RC101	8	5	6	3	2	4	381.15	181.3	381.15	0.00%	152.0
RC101	8	5	10	3	4	3	377.25	3600.0	377.25	0.00%	187.5
RC101	8	6	5	3	2	4	446.8	3600.0	446.8	0.00%	232.4
RC101	8	6	6	3	2	4	461.27	1191.8	461.27	0.00%	126.9
RC101	8	6	10	3	4	3	446.8	3600.1	446.8	0.00%	250.2
RC101	8	7	5	3	2	3	373.2	3600.0	373.34	0.04%	356.8
RC101	8	7	6	3	2	3	373.2	3600.0	373.2	0.00%	359.5
RC101	8	7	10	3	4	3	373.2	3600.0	373.2	0.00%	507.0
R203	10	5	5	3	2	4	334.69	3600.0	334.69	0.00%	302.9
R203	10	5	6	3	2	4	334.69	87.2	334.69	0.00%	312.3
R203	10	5	10	3	4	4	334.69	3600.1	334.69	0.00%	317.3
R203	10	6	5	3	2	4	374.68	3600.0	374.68	0.00%	394.4
R203	10	6	6	3	2	4	329.51	3600.0	339.02	2.89%	327.8
R203	10	6	10	3	4	4	373.27	3600.0	371.48	-0.48%	374.9
R203	10	7	5	3	2	4	394.71	3600.1	374.93	-5.01%	395.5
R203	10	7	6	3	2	4	384.1	731.6	384.1	0.00%	376.5
R203	10	7	10	3	4	4	385.34	3600.1	392.73	1.92%	387.2
C203	10	5	5	3	2	4	287.74	3600.0	287.74	0.00%	253.9
C203	10	5	6	3	2	4	285.77	2515.8	285.77	0.00%	205.5
C203	10	5	10	3	4	2	272.08	3600.0	270.33	-0.64%	245.1
C203	10	6	5	3	2	4	342.02	3600.0	346	1.16%	252.1
C203	10	6	6	3	2	4	340.85	3600.0	340.85	0.00%	251.7
C203	10	6	10	3	4	5	331.83	3600.1	335.42	1.08%	450.6
C203	10	7	5	3	2	4	371.24	3600.0	375.76	1.22%	177.0
C203	10	7	6	3	2	4	371.1	3235.5	385.07	3.76%	179.6
C203	10	7	10	3	4	3	370.98	3600.1	367.73	-0.88%	238.0
RC103	10	5	5	3	2	4	510.09	3600.0	516.78	1.31%	307.1
RC103	10	5	6	3	2	5	510.09	3600.1	489.7	-4.00%	267.3
RC103	10	5	10	3	4	5	510.09	3600.1	489.7	-4.00%	404.8
RC103	10	6	5	3	2	3	395.93	3600.0	388.97	-1.76%	173.4
RC103	10	6	6	3	2	4	423.59	3600.0	423.59	0.00%	149.8
RC103	10	6	10	3	4	2	383.24	3600.0	383.24	0.00%	166.7
RC103	10	7	5	3	2	4	564.95	3600.1	566.86	0.34%	238.4
RC103	10	7	6	3	2	4	526.5	3600.0	526.5	0.00%	231.9
RC103	10	7	10	3	4	5	530.66	3600.1	540.84	1.92%	362.0
C101	12	5	5	3	2	7	*inf.*	3600.0	497.88		468.9
C101	12	5	6	3	2	7	483.25	3600.1	469.4	-2.87%	480.6
C101	12	5	10	3	4	5	*inf.*	3600.1	485.71		590.4
C101	12	6	5	3	2	4	353.39	3600.0	352.35	-0.29%	334.2
C101	12	6	6	3	2	7	343.04	3600.1	343.04	0.00%	428.8
C101	12	6	10	3	4	4	341.89	3600.1	341.39	-0.15%	446.3
C101	12	7	5	3	2	2	166.79	3600.0	167.84	0.63%	239.7
C101	12	7	6	3	2	3	164.95	3600.0	164.52	-0.26%	279.7
C101	12	7	10	3	4	3	169.59	3600.1	161.86	-4.56%	369.4
R103	12	5	5	3	2	8	500.9	3600.0	478.9	-4.39%	418.2
R103	12	5	6	3	2	6	442.01	3600.1	443.67	0.38%	290.2
R103	12	5	10	3	4	6	*inf.*	3600.1	442.01		460.1
R103	12	6	5	3	2	4	362.08	3600.0	370.97	2.46%	297.9
R103	12	6	6	3	2	6	*inf.*	3600.1	365.27		383.8

Continued on next page

Instance Details							CPLEX		ALNS		
Name	**n**	**S**	**|W|**	**Q**	**D**	**T**	**$z(s^*_C)$**	**CPU (secs.)**	**$z(s^*_A)$**	**Gap %**	**CPU (secs.)**
R103	12	6	10	3	4	5	*inf.*	3600.1	353.56		346.0
R103	12	7	5	3	2	7	468.67	3600.0	473.17	0.96%	374.9
R103	12	7	6	3	2	5	464.11	3600.0	464.11	0.00%	289.8
R103	12	7	10	3	4	6	464.11	3600.1	468.61	0.97%	493.3
RC201	12	5	5	3	2	4	573.96	3600.1	558.55	-2.68%	275.7
RC201	12	5	6	3	2	4	520.2	3600.0	473.95	-8.89%	297.9
RC201	12	5	10	3	4	3	599.98	3600.1	551.17	-8.14%	285.8
RC201	12	6	5	3	2	4	473.25	3600.0	474.77	0.32%	228.7
RC201	12	6	6	3	2	6	*inf.*	3600.0	474.77		289.6
RC201	12	6	10	3	4	3	495.35	3600.1	457.46	-7.65%	224.7
RC201	12	7	5	3	2	4	536.79	3600.0	533.52	-0.61%	244.6
RC201	12	7	6	3	2	4	531.53	3600.0	554.08	4.24%	200.3
RC201	12	7	10	3	4	4	*inf.*	3600.0	498.98		324.9
C201	15	5	5	3	2	4	*inf.*	3600.1	466.73		381.4
C201	15	5	6	3	2	4	437.64	3600.0	433.29	-0.99%	396.7
C201	15	5	10	3	4	5	*inf.*	3600.2	443.02		579.2
C201	15	6	5	3	2	6	*inf.*	3600.1	581.49		313.4
C201	15	6	6	3	2	7	*inf.*	3600.0	703.51		260.0
C201	15	6	10	3	4	3	*inf.*	3600.1	824.69		207.6
C201	15	7	5	3	2	7	556.55	3600.1	574.98	3.31%	409.7
C201	15	7	6	3	2	7	550.08	3600.0	549.65	-0.08%	536.1
C201	15	7	10	3	4	7	*inf.*	3600.1	597.56		816.2
R201	15	5	5	3	2	4	*inf.*	3600.0	526.11		254.3
R201	15	5	6	3	2	4	540.06	3600.0	520.77	-3.57%	249.8
R201	15	5	10	3	4	3	*inf.*	3600.1	494.6		298.2
R201	15	6	5	3	2	8	*inf.*	3600.1	550.88		570.5
R201	15	6	6	3	2	7	546.55	3600.1	555.5	1.64%	534.1
R201	15	6	10	3	4	4	*inf.*	3600.1	546.56		457.4
R201	15	7	5	3	2	8	595.16	3600.0	599.48	0.73%	635.4
R201	15	7	6	3	2	7	554.65	3600.1	564.66	1.80%	642.4
R201	15	7	10	3	4	6	*inf.*	3600.1	637.83		726.9
RC203	15	5	5	3	2	6	*inf.*	3600.0	762.97		417.8
RC203	15	5	6	3	2	5	681.42	3600.0	677.39	-0.59%	304.0
RC203	15	5	10	3	4	5	*inf.*	3600.2	749.18		415.3
RC203	15	6	5	3	2	8	*inf.*	3600.0	776.73		739.3
RC203	15	6	6	3	2	10	*inf.*	3600.0	827.06		665.2
RC203	15	6	10	3	4	6	*inf.*	3600.2	755.89		696.5
RC203	15	7	5	3	2	8	724.65	3600.0	739.02	1.98%	461.3
RC203	15	7	6	3	2	7	730.65	3600.0	720.22	-1.43%	469.2
RC203	15	7	10	3	4	5	*inf.*	3600.2	752.48		484.8
Average								3,224.8		-0.24%	336.2

$z(s^*_C)$ denotes the value of the best solution found by CPLEX.
$z(s^*_A)$ denotes the value of the best solution found by ALNS.
Gap % is computed as $100 \times \frac{(z(s^*_A) - z(s^*_C))}{z(s^*_C)}$.
inf. indicates those cases where CPLEX did not find any feasible solution within the time limit.

Table 7: Small-size instances: Detailed computational results.