algorithms

MDPI

Article

Matheuristics and Column Generation for a Basic Technician Routing Problem

Nicolas Dupin [1,*], Rémi Parize [2] and El-Ghazali Talbi [3]

[1] Université Paris-Saclay, Laboratoire Interdisciplinaire des Sciences du Numérique (LISN), 91405 Orsay, France
[2] Independent Researcher, 75509 Paris, France; remi.parize@polytechnique.org
[3] CNRS UMR 9189-CRIStAL-Centre de Recherche en Informatique Signal et Automatique de Lille, Université Lille, F-59000 Lille, France; el-ghazali.talbi@univ-lille.fr
* Correspondence: nicolas.dupin@universite-paris-saclay.fr

Abstract: This paper considers a variant of the Vehicle Routing Problem with Time Windows, with site dependencies, multiple depots and outsourcing costs. This problem is the basis for many technician routing problems. Having both site-dependency and time window constraints lresults in difficulties in finding feasible solutions and induces highly constrained instances. Matheuristics based on Mixed Integer Linear Programming compact formulations are firstly designed. Column Generation matheuristics are then described by using previous matheuristics and machine learning techniques to stabilize and speed up the convergence of the Column Generation algorithm. The computational experiments are analyzed on public instances with graduated difficulties in order to analyze the accuracy of algorithms for ensuring feasibility and the quality of solutions for weakly to highly constrained instances. The results emphasize the interest of the multiple types of hybridization between mathematical programming, machine learning and heuristics inside the Column Generation framework. This work offers perspectives for many extensions of technician routing problems.

Keywords: optimization; operations research; mathematical programming; mixed integer linear programming; Dantzig–Wolfe decomposition; column generation; matheuristics; hybrid heuristics; parallel algorithms; workforce scheduling and routing; vehicle routing problems

Citation: Dupin, N.; Parize, R.; Talbi, E.-G. Matheuristics and Column Generation for a Basic Technician Routing Problem. *Algorithms* **2021**, *14*, 313. https://doi.org/10.3390/a14110313

Academic Editor: Frank Werner

Received: 1 September 2021
Accepted: 25 October 2021
Published: 27 October 2021

1. Introduction

If mathematical programming and especially Mixed Integer Linear Programming (MILP) are powerful frameworks for modeling a vast diversity of $\mathcal{NP}$-hard combinatorial optimization problems, including complex real-world optimization problems, the resolution with exact methods such as the Branch and Bound (B&B) algorithm is limited in practice for large instances of real-world applications [1]. Metaheuristics find good solutions of large optimization problems without optimality or quality proofs. Highly constrained instances of optimization problems are a major bottleneck of metaheuristics. Metaheuristics and mathematical programming approaches have complementary advantages and drawbacks, and it results in their hybridization, called *matheuristics* [2,3]. Some matheuristics use exact methods to solve large neighborhoods of an exponential size, which tends to have fewer local minimums of a better quality than by using small neighborhoods [2,4]. Solving the exact neighborhoods allows studying the quality of implied local minimums; such intermediate results are of interest for selecting which types of neighborhoods have to be carefully implemented in a local search heuristic [4]. Matheuristics allow designing many variants of heuristic operators; it takes profit of parallel implementation techniques [5]. Having also other complementary advantages and drawbacks, machine learning (ML) techniques are also recently considered in such hybridization schemes [6,7]. Note that such hybridization can also provide dual bounds for large instances of optimization problems, using aggregation or decomposition techniques to compute dual bounds for heuristically

reduced or decomposed problems with a proof that a dual bound of the original problem is computed despite the heuristic reduction [8,9]. ML techniques are useful in this context for selecting the promising dual bound among many alternatives [9].

Such methodology is investigated for a variant of the Vehicle Routing Problem with Time Window (VRPTW), seeking to service customers in defined time windows with a fleet of vehicles. VRPTW is a widely studied problem in the operations research literature due to its numerous applications in transportation, distribution and logistics, with significant academic works in MILP formulations and metaheuristics [10]. Among the exact methods, Column Generation (CG) approaches are efficient for a large class of vehicle routing problems and are actively studied nowadays [11,12]. CG approaches can be extended to CG heuristics or used in any hybrid heuristic to tackle large instances of vehicle routing problems [13–15]. Recently, the contribution of ML techniques for solving CG sub-problems is studied for some vehicle routing problems [16,17].

This paper considers a variant of the VRPTW which is a common basis for many technician routing problems, as in [18,19]. Site-dependency constraints model that customers can be served by only a subset of vehicles, which is related to the skills of technicians to engage in interventions. Contrary to VRPTW problems where each request must be served, we consider the possibility of realizing a subset of interventions by considering outsourcing costs. We consider, furthermore, multi-depot constraints: Technicians are starting and finishing their tours in different depots. Matheuristics are investigated and compared to local search and exact approaches. New instances are proposed with relations between subsets of instances to analyze the impact of highly to weakly constrained instances in the solving capabilities of different optimization algorithms. This paper investigates how to design hybrid algorithms to solve this optimization problem efficiently.

The remainder of this paper is structured as follows. Section 2 provides an overview of the problem statement and its state of the art. Section 3 presents two types of compact MILP formulations for the problem. Section 4 designs constructive and local search matheuristics by using compact MILP formulations. Section 5 investigates the hybridization of matheuristics with CG techniques for an extended MILP formulation. The computational results are analyzed in Section 6. In Section 7, the conclusions of this works are highlighted, drawing the perspectives raised by our contributions.

2. Problem Statement and Related Work

This section defines our technician routing problem, situates the closest problems from the literature and presents state of the art methods for these related problems, focusing particularly on mathematical programming and meta-heuristic approaches and their hybridization.

2.1. Problem Statement and Notation

The variant of VRPTW considered in this paper, namely TRPTW for Technician Routing Problem with Time Windows, is described by using the terminology from [18]. The routing optimization concerns a set of I vehicles with J requests and denoted jobs or interventions, in different locations of customers. Interventions require the specific skills of technicians and have time window constraints on the arrival time of the technicians. In the following developments, a technician denotes equivalently a vehicle, as if each vehicle is driven by a single technician, and interventions require only one technician with the required qualifications. One may consider equivalently a technician team, which is defined a priori in the optimization, sharing the same vehicle and working together in the interventions, and there is no splitting of a technician team for realizing different interventions at the same time. The resulting skills of the technician team are mathematically equivalent to a skill set and a subset of interventions that can be realized, as if we have a fictive single technician with these resulting skills. For the sake of simplifying the presentation, the following developments consider one technician by vehicle, with skills associated to

this technician. Furthermore, technicians are starting and finishing their tour in defined depots, with time-window constraints for leaving and going back to the depot.

It is possible to plan only a subset of interventions; a penalty in the objective function is associated for each request that is not served. The objective function to minimize is the total length of vehicles' routes, adding penalties for the interventions that are not realized. Such objective function may be observed as a weighted sum optimization, considering the bi-objective minimization of routing costs and unserved requests. Here, a weighted sum optimization makes sense with relevant values of penalties. Some interventions may be outsourced and realized by other companies, and penalties correspond to outsourcing costs. Some interventions may be urgent, and they are preferred be realized within the time horizon with high penalty, whereas other interventions can be postponed for another day and have low penalties. Having high and equal penalties for jobs is equivalent to a hierarchical optimization that firstly maximizes the number of jobs that are planned and then minimizing the route lengths. Mathematically, it is feasible for the optimization problem to have empty routes and only penalization. However, finding routes that serve all the requests is a NP-hard decision problem.

The notation for sets, indexes and input parameters is gathered in Table 1. The jobs are indexed with $j \in \mathcal{J} = [\![1, J]\!]$. The intervention of job j has to begin in time window $[T_j^{min}, T_j^{max}]$, its duration is L_j and its outsourcing cost is P_j. For the sake of unifying notation, we indexed technicians with $i \in \mathcal{I} = [\![J+1, J+I]\!]$ and defined nodes $n \in \mathcal{N} = \mathcal{I} \cup \mathcal{J} = [\![1, J+I]\!]$ to index the different locations of jobs and depots for departure and arrival of technicians' routes. $D_{n,n'}$ and $T_{n,n'}$ denote, respectively, the kilometric distance and transportation time from the location of node n to its of node n'. Note that distances are calculated with Euclidean rules so that we have triangle inequalities $D_{n,n'} + D_{n',n''} \geqslant D_{n,n''}$ and $T_{n,n'} + T_{n',n''} \geqslant T_{n,n''}$. Technician $i \in \mathcal{I}$ starts from his depot after T_i^{start} and must be back at his depot before T_i^{end}. $\mathcal{J}_i$ denotes the subset of jobs that technician i can complete.

Table 1. Notation, sets, indexes and input parameters.

I	Number of technicians (or equivalently vehicles).
J	Number of jobs (or equivalently customers, requests, or request location).
$n \in \mathcal{N} = [\![1, J+I]\!]$	Index and set for nodes, a node is a technician depot or a job location.
$j \in \mathcal{J} = [\![1, J]\!]$	Index and set for customers, the jobs for the technicians.
$i \in \mathcal{I} = [\![J+1, J+I]\!]$	Index and set for technicians.
$\mathcal{J}_i$	Subset of jobs that technician $i \in \mathcal{I}$ can complete with skill constraints.
$\mathcal{N}_i = \mathcal{J}_i \cup \{i+J\}$	Subset of nodes that technician $i \in \mathcal{I}$ can visit (depot and skill constraints).
L_j	Duration (amount of time) of job $j \in \mathcal{J}$.
P_j	Cost penalization if job $j \in \mathcal{J}$ is not planned (or outsourcing cost).
K_i	Upper bound for the number of jobs that technician $i \in \mathcal{I}$ can process.
$D_{n,n'}$	Distance from the location of node $n \in \mathcal{N}$ to the location of node $n' \in \mathcal{N}$.
$T_{n,n'}$	Transportation time from the location of node $n \in \mathcal{N}$ to the location of node $n' \in \mathcal{N}$.
$[T_i^{start}, T_i^{end}]$	Working time window for technician $i \in \mathcal{I}$, to leave and come back to the depot.
$[T_j^{min}, T_j^{max}]$	Time window for the beginning of job $j \in \mathcal{J}$.

N.B: $[\![a, b]\!] = [a, b] \cap \mathbb{Z}$ denotes integer intervals in this table.

2.2. Related Problems in the Literature

2.2.1. Related Vehicle Routing Problems

Some classical variants of VRPTW are sub-cases of TRPTW. In the instances of the literature, the VRPTW considers complete graphs: Each vehicle can visit each city [20]. The simplest variant of VRPTW considering that some jobs can be realized by only a subset

of vehicles is the *Site-dependent* VRPTW (SDVRPTW) [21]. The extension of the VRPTW considering several depots is known as *Multi-depot* VRPTW (MDVRPTW) [22]. There is one major difference between TRPTW and a site dependent MDVRPTW: contrary to VRPTW extensions, it is feasible to serve any subset of the requests for TRPTW by considering outsourcing costs, i.e., penalties for interventions that are not realized. MDVRPTW and SDVRPTW are sub-problems of TRPTW considering the same transportation graph, high and equal outsourcing costs for jobs and upper bounds of any route visiting each customer (which can be calculated easily): If the optimal cost of such TRPTW is greater than the upper bound, the original MDVRPTW or SDVRPTW is not feasible; otherwise, we have an optimal solution visiting each customer. We note that such outsourcing costs or equivalently fictive vehicles with the penalization cost are used by CG approaches for VRPTW problems to ensure feasibility for CG iterations [23].

Skill VRPTW is another variant of the VRPTW inspired by technician routing applications [24,25]. Skills model the ability of vehicles (or technicians) to proceed to specific locations (or intervention with specific qualification). Skills are equivalent to site-dependencies, and skills VRPTW and SDVRPTW have the same set of feasible solutions. Skill VRPTW is an extension of SDVRPTW, and costs of vehicle usage are increasing with skills, whereas costs depend only on traveled distance for SDVRPTW. Note that Heterogeneous Fleet VRPTW (HFVRPTW), where vehicles are differentiated in the travel time and costs, is a common extension for skills VRPTW and SDVRPTW [26]. Indeed, site-dependent constraints can be considered with infinite costs (or a trivial upper bound of any feasible solution) in a HFVRPTW. For the sake of having realistic solutions with real-world application, optimal routes in skill VRPTW can be unbalanced; some works consider load balancing in the objective function [27]. This results in a multi-objective extension and minimizing routing costs conjointly with the maximization of route balance for better equity and robustness relative to uncertainty [28]. Recently, an extension of skill VRPTW was proposed by taking into account that intervention time depends on the skill levels [29].

2.2.2. Related Technician Routing Problems

TRPTW can be observed as a simple case of many technician routing problems. A large diversity of technician routing problems was studied, with many variants of additional constraints [30]. The technician routing and scheduling problem (TRSP) and workforce scheduling and routing problem (WSRP) denote such extensions of TRPTW [31,32].

A rich technician routing problem was proposed for the ROADEF Challenge 2007, organized by the French Operations Research society (ROADEF) [18]. This technician routing problem optimizes the daily assignment of technicians into teams, of teams to tasks and of teams to daily routes in a multi-period problem; site-dependency is implied by the teams composition. There are also precedence constraints among interventions; for instance, some interventions must be preceded by the delivery of some equipment. The TRSP considered by [31,32] considers a single period version of the ROADEF Challenge 2007 by analyzing the impact of team building optimization. Recently, the VeRoLog Solver Challenge (VSC) 2018–2019 proposed a rich technician routing and scheduling problem with truck routing and team building optimization [19]. VSC differs from the ROADEF Challenge 2007 with additional scheduling constraints for technicians. For instance, a single technician may perform at most one tour per day and must not perform tours on more than five consecutive days.

In the TRSP considered by [33], vehicles are equivalent to technicians with no team building optimization, which defines site-dependencies. This TRSP includes additional request requirements for spare parts and tools that can be seen as non-renewable resources. Technicians collect tools and spare parts from a central depot at any time during the execution of their routes. Recently, an extension was provided considering a mix of conventional and electric vehicles, which requires considering recharging constraints [34].

Multi-period and dynamic technician routing problems consider many variants of scheduling constraints. The technician routing problem with experience-based service

times addresses the possibility that technicians learn and improve their skills with an impact on performance [35]. Dynamic versions correspond to realistic situations where new requests, including emergency interventions, arise during the intervention tour and requires re-optimization [36,37]. Urgent interventions may induce considering the minimization of service disruption over a time horizon, which is different from standard VRPTW objective function [38].

2.3. Algorithms for Solving Related Problems

2.3.1. Solving Related Vehicle Routing Problems

Vehicle routing problems such as SDVRPTW and MDVRPTW are known to be efficiently solved by optimality using CG techniques and Branch and Price (B&P) or Branch, Cut and Price (B&C&P) [11]. Recent trends solve a large class of vehicle routing problems in a generic way [12]. A crucial point is to solve CG sub-problems as efficiently as possible, with significant works solving the shortest path problems with resource constraints with labelling algorithms [39,40]. However, very few specific works concerned the application of the CG algorithm for the SDVRPTW variant. SDVRPTW was recently studied in unified B&C&P applying for HFVRPTW [26]. Specific works in mathematical programming concerned HFVRP [41]. A unified and efficient approach was provided with the dual ascent method with Lagrangian bounds for HFVRP, SDVRP and MDVRP [42]. A B&C&P algorithm for the multi-depot HFVRPTW was provided by [43].

Metaheuristics are widely used to solve vehicle routing problems [44]. As with exact methods, recent trends develop generic heuristics for a unified formulation of several extensions of VRPTW. A unified formulation of MDVRPTW and SDVRPTW is given by [20]. Such unified extension was solved efficiently firstly with tabu search [20], secondly with Adaptive Large Neighborhood Search (ALNS) [45] and lastly with a hybrid Genetic Algorithm (GA) with adaptive diversity management [46]. Ant Colony Optimization (ACO) is another population metaheuristic that was proven efficient for some vehicle routing problems related to technician routing [47].

Additional constraints in rich VRPTW induce difficulties for metaheuristics, and matheuristics are widely used in such contexts, especially CG matheuristics [14]. The authors of [15] provide a taxonomy of matheuristics for vehicle routing problems: *Decomposition approaches* are constructive heuristics solving iteratively sub-problems as in [48]; *improvement heuristics* apply local search heuristics with large MILP neighborhoods as in [49]; and *CG-based heuristics* derive primal heuristics from Lagrangian relaxations or CG and B&P algorithms as in [13,50]. The hybridization of ML techniques with CG algorithm was proven efficient for some vehicle routing problems [16,17].

2.3.2. Solving Technician Routing Problems

Efficient algorithms for solving technician routing problems are similar to the ones for VRPTW extensions. Since recently, technician routing problems were mostly solved by heuristics. An ALNS algorithm was proposed to solve a TRSP as a SDVRPTW extension [31]. A simple Iterated Local Search (ILS) has been proposed for the same problem and showed excellent results in short solving time [32]. Population meta-heuristics showed also excellent results for technician routing problems. Multi-attribute VRPTW is an extension that is also a basis for technician routing problems; hybrid GAs were shown to be efficient [51]. A Particle Swarm Optimization (PSO) algorithm was proven efficient for solving a TRSP [52]. An ACO heuristic was designed to solve efficiently a multi-period workforce scheduling and routing problem with dependent tasks [53].

Recent progresses in MILP solving allow considering exact methods for technician routing problems with CG algorithms, B&P and B&C&P approaches [54–56]. In the previously mentioned TRP with site-dependencies, two approaches considered a matheuristic using an MILP set covering formulations [33,57]. In such cases, an MILP computation selects routes among routes that are calculated by heuristics, similarly with the master problem of the CG algorithm. In such cases, the columns are generated by using only

heuristics and without using reduced costs, a parallel ALNS for [33] and an hybridization of ALNS and Variable Neighborhood Descent (VND) for [57].

Due to the large diversity of specific technician routing problems that were studied [30], we focus in the following on algorithms solving VSC 2018–2019 [19] and the ROADEF Challenge 2007 [18]. For such problems, several studies deal with exactly the same optimization problem and public instances; it allows relevant comparisons. The winner of VSC 2018–2019 used a trajectory local search heuristic, which is a hybridization of ALNS and Variable Neighborhood Search (VNS) [58]. A similar quality of results was obtained with a set partitioning matheuristic "Columnwise neighborhood search" [59]. Competitive results were also obtained with a population hyper-heuristic [60] and by using an iterated local search using several neighborhood search operators and destroy/repair heuristics [61].

The winner of the ROADEF Challenge 2007 considered a three phase matheuristic [48]. A preprocessing phase estimates the number of technicians required in each team through the first MILP. Then, a second MILP computes a lower bound allowing job preemption. Multiple solutions are constructed by iteratively solving a third MILP by matching formulation with different parameter values but without introducing randomness. This approach was improved after by [62]. Another two-step decomposition matheuristic provided good results and is ranked fifth [63]. Trajectory heuristics provided also excellent results, with ranks 2, 3 and 4, local search with multiple neighborhoods and operators for [64], ALNS for [65] and Greedy randomized adaptive search procedure (GRASP) for [66].

3. MILP Compact Formulations

In this section, two alternative MILP formulations are provided for TRPTW, with cuts to improve the quality of the Linear Programming (LP) relaxation for a better efficiency of B&B tree search.

3.1. First MILP Formulations

We define binary variables $u_{i,n,n'} \in \{0,1\}$ for each technician $i \in \mathcal{I}$ and each pair of nodes $(n,n') \in \mathcal{N}^2$, where $u_{i,n,n'} = 1$ if and only if technician i proceeds to node n' immediately after node n. For each technician i, such definition unifies routes between jobs locations $u_{i,j,j'}$ for each pair of jobs $(j,j') \in \mathcal{J}^2$, a route between depot and job location, respectively, $u_{i,i,j} = 1$ and $u_{i,j,i} = 1$, is defined to perform intervention $j \in \mathcal{J}$ as the first or last job of the day and $u_{i,i,i} = 1$ is the case where technician i has no job in his working day. With such a definition, we need only to consider variables $u_{i,n,n'}$ for each technician $i \in \mathcal{I}$ and each pair of nodes $(n,n') \in (\mathcal{J}_i \cup \{i\})^2 = \mathcal{N}_i^2$, and the other variables are zeros with skill constraints and definition of departure and arrival depots. Furthermore, we introduce continuous variables t_j by representing the start time for job j, bounded with the time-windows constraints.

It results in the following MILP compact formulation below. The objective (1) for minimizing is composed of two linear expressions of variables $u_{i,n,n'}$, the total length of the routes and the penalties for postponed or outsourced jobs j having zero values for variables $u_{i,n,j}$ for all $i \in \mathcal{I}$ and $n \in \mathcal{N}$. Constraints (3) are implied by the definition of variables, $u_{i,j,j} = 0$ for each technician $i \in \mathcal{I}$ and each job $j \in \mathcal{J}$. Constraints (2) are elementary constraints: Each intervention is realized at most one; it can be postponed or outsourced if variables $u_{i,n,j}$ are zeros for $i \in \mathcal{I}$ and $n \in \mathcal{N}$. With triangle inequality, it is sub-optimal to visit a customer twice or more times. Indeed, having triangle inequalities $D_{n,n'} + D_{n',n''} \geqslant D_{n,n''}$ and $T_{n,n'} + T_{n',n''} \geqslant T_{n,n''}$, removing jobs planned twice, is still a feasible solution, and it decreases the objective function; there is an optimal solution allocating exactly one technician per job. Elementarity constraints are necessary here in the MILP model as multiple visits induce a negative penalization and, thus, a high bonus in the objective function. Constraints (4) are flow conservation constraints for modelling routes for technicians: the flow leaving and the flow arriving to a node for technician i is the same flow that is leaving from the node. Constraints (5) initialize flow constraints;

there is exactly one arc leaving from a depot, and it can be $u_{i,i,i} = 1$ in the case of an empty route for technician i so that, in any case, the leaving flow is exactly one. Note that constraints (4) and (5) imply that $\sum_{n \in \mathcal{N}_i} u_{i,n,i} = 1$ for all technician i, which is a one value flow for going back to the depot; such constraints are not necessary. Constraints (2) can be seen as flow capacity constraints. Constraints (6)–(8) express time windows constraints, respectively, between two consecutive jobs and for the starting and finishing time of technicians. With constraints (6)–(8), the formulation does not require any sub-tour constraints. Coefficients $M^1_{j,j'}$, $M^2_{i,j}$ and $M^3_{i,j}$ can be chosen as follows: $M^1_{j,j'} = T^{max}_j + L_j + T_{j,j'} - \tilde{t}^{min}_{j'}$, $M^2_{i,j} = T^{start}_i + T_{i,j} - \tilde{t}^{min}_j$ and $M^3_{i,j} = T^{max}_j + L_j + T_{i,j} - T^{end}_i$.

$$\min_{u_{i,n,n'}, t_j} \quad \sum_{i \in \mathcal{I}} \sum_{n,n' \in \mathcal{N}_i} D_{n,n'} u_{i,n,n'} + \sum_{j \in \mathcal{J}} P_j \left(1 - \sum_{i \in \mathcal{I}} \sum_{n \in \mathcal{N}_i} u_{i,j,n} \right) \tag{1}$$

$$\forall n \in \mathcal{N}_i, \qquad \sum_{i \in \mathcal{I}} \sum_{n' \in \mathcal{N}_i} u_{i,n,n'} \leqslant 1 \tag{2}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \qquad u_{i,j,j} \leqslant 0 \tag{3}$$

$$\forall i \in \mathcal{I}, \forall n \in \mathcal{N}_i, \qquad \sum_{n' \in \mathcal{N}_i} u_{i,n,n'} = \sum_{n' \in \mathcal{N}_i} u_{i,n',n} \tag{4}$$

$$\forall i \in \mathcal{I}, \qquad \sum_{n \in \mathcal{N}_i} u_{i,i,n} = 1 \tag{5}$$

$$\forall j, j' \in \mathcal{J}, \qquad t_j + L_j + T_{j,j'} \leqslant t_{j'} + \left(1 - \sum_{i \in \mathcal{I}} u_{i,j,j'} \right) \cdot M^1_{j,j'} \tag{6}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \qquad T^{start}_i + T_{i,j} \leqslant t_j + \left(1 - u_{i,i,j} \right) \cdot M^2_{i,j} \tag{7}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \qquad t_j + L_j + T_{i,j} \leqslant T^{end}_i + \left(1 - u_{i,j,i} \right) \cdot M^3_{i,j} \tag{8}$$

$$\forall i \in \mathcal{I}, \forall n, n' \in \mathcal{N}_i \qquad u_{i,n,n'} \in \{0,1\}, t_j \in [T^{min}_j, T^{max}_j] \tag{9}$$

Infeasibilities due to time windows can written directly on variables $u_{i,j,j'}$ for $j, j' \in \mathcal{J}_i$ by following rules allowing the removal of variables that could be non zeros in the LP relaxation. If $T^{min}_j + T_{j,j'} + L_j > T^{max}_j$, then $u_{i,j,j'} = 0$; it is not possible to reach intervention j' on time after having realized intervention j. If $T^{start}_i + T_{i,j} > T^{max}_j$, it is not possible to reach location j on time by starting from depot i, then $u_{i,i,j} = 0$. If $T^{min}_j + T_{i,j} + L_j > T^{end}_i$, then $u_{i,j,i} = 0$; it is not possible to reach depot i on time if j is realized by technician i. For both previous cases, it is actually unfeasible with triangle inequalities that technician i realizes intervention j; all the variables $u_{i,j,n}$ and $u_{i,n,j}$ can be removed.

If constraints (6)–(8) define integer solutions, such constraints are known to induce LP relaxations of a poor quality [67]. In the following, cuts are provided to improve significantly dual bounds without increasing too much the computation time for the dual bounds. Sub-tour cuts, such as in the Traveling Salesman Problem (TSP), can cut continuous solutions induced by the "big M" constraints. One can forbid firstly the sub-tours between two jobs j and j' with constraints $u_{i,j,j'} + u_{i,j',j} \leqslant 1$. Such cuts can be lifted, observing that cases $u_{i,j,j'} = u_{i,j',j} = 0$ are feasible here when the vehicle is not leaving the depot. It induces tighter cuts.

$$\forall i \in \mathcal{I}, \forall j < j' \in \mathcal{J}_i, \quad u_{i,j,j'} + u_{i,j',j} \leqslant 1 - u_{i,i,i} \tag{10}$$

Note that it is possible to have $u_{i,j,i} = u_{i,i,j} = 1$ when a technician does a single intervention in the route, and previous cuts cannot be extended for $j, j' \in \mathcal{N}_i$.

3.2. Four-Index MILP Formulation with Ordinality

Many alternative MILP formulations can be designed with the formulation variants of sub-tours in TSP as analyzed in [68]. Similarly to the formulation of Fox–Gavish–Graves

(FGG) [69], variables can be modeled with 4-index variables $z_{i,j,n,k} \in \{0,1\}$, such that $z_{i,j,n,k} = 1$ if and only if the k-th job ($k \in [\![1, K_i]\!]$) that is operated by technician i is job $j \in \mathcal{J}_i$ and then the job (of depot) $n \in \mathcal{N}_i$ is reached after intervention j. Without time window constraints and without K_i upper bound on the number of intervention per route, this would induce a number of variables in $O(\mathcal{I} \times \mathcal{J}^3)$ instead of $O(\mathcal{I} \times \mathcal{J}^2)$ for the last MILP formulation. A first, the issue will be to analyze if the better quality LP relaxation compensates the increasing size of the model. The maximum number of interventions in a day K_i allows having $O(K \times \mathcal{I} \times \mathcal{J}^2)$ variables, with $K = \max_i K_i$. With the variables of the previous formulation, there is the following relation.

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \forall n \in \mathcal{N}_i, \quad u_{i,j,n} = \sum_{k=1}^{K_i} z_{i,j,n,k} \tag{11}$$

The following MILP formulation models TRPTW with variables $z_{i,j,n,k}$ is as follows.

$$\min_{z,t} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \sum_{n\in\mathcal{N}_i} \sum_{k=1}^{K_i} D_{j,n} z_{i,j,n,k} + \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \sum_{n\in\mathcal{N}_i} D_{i,j} z_{i,j,n,1} + \sum_{j\in\mathcal{J}} P_j \left(1 - \sum_{i,j',k} z_{i,j,n,k}\right) \tag{12}$$

$$\forall j \in \mathcal{J}, \quad \sum_{n\in\mathcal{N}_i} \sum_{k=1}^{K_i} \sum_{i\in\mathcal{I}} z_{i,j,n,k} \leqslant 1 \tag{13}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \quad \sum_{k=1}^{K_i} z_{i,j,j,k} \leqslant 0 \tag{14}$$

$$\forall i \in \mathcal{I}, \quad \sum_{j\in\mathcal{J}_i} \sum_{n\in\mathcal{N}_i} z_{i,j,n,1} \leqslant 1 \tag{15}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \forall k \in [\![1, K_i - 1]\!], \quad \sum_{j'\in\mathcal{J}_i} z_{i,j',j,k} = \sum_{n\in\mathcal{N}_i} z_{i,j,n,k+1} \tag{16}$$

$$\forall j, j' \in \mathcal{J}, \quad t_j + L_j + T_{j,j'} \leqslant t_{j'} + \left(1 - \sum_{k=1}^{K_i} \sum_{i\in\mathcal{I}: j\in\mathcal{J}_i} z_{i,j,j',k})\right) \cdot M^1_{j,j'} \tag{17}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \quad T_i^{start} + T_{i,j} \leqslant t_j + \left(1 - \sum_{n\in\mathcal{N}_i} z_{i,j,n,1}\right) \cdot M^2_{i,j} \tag{18}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \quad t_j + L_j + T_{i,j} \leqslant T_i^{end} + \left(1 - \sum_{k=1}^{K_i} z_{i,j,i,k}\right) \cdot M^3_{i,j} \tag{19}$$

$$\forall j \in \mathcal{J}, \quad t_j \in [T_j^{min}, T_j^{max}] \tag{20}$$

Note that the objective function is in three parts, and $D_{j,n} z_{i,j,n,k}$ terms do not count the distance from the depot to the first jobs of technicians. Constraints (13) are straightforwardly equivalent to the previous elementarity constraints (2). Constraints (14) are implied by the definition of variables, $z_{i,j,j,k} = 0$ for each technician $i \in \mathcal{I}$, $j \in \mathcal{J}_i$ and $k \in [\![1, K_i]\!]$. Constraints (15) initialize flow constraints, as in (5); the difference is that flow initialization is written after the first job, and this flow is not anymore equal to one as there are no variables indicating straightforwardly if an empty route is chosen. In such case, all the variables related to the technician are zeros. Constraints (16) are the equivalent of previous flow constraints (4) for modeling routes for technician; the balance of flow is here considered regarding the ordinality k. Constraints (17)–(19) express time windows constraints, respectively, between two consecutive jobs and for starting and finishing time of technicians, similarly with constraints (6)–(8). Constraint (17) is the same constraint set as (6) by using relations (11). Constraint (18) is equivalent with (7), observing that j is the first intervention of technician i if and only if there exist $n \in \mathcal{N}_i$ such that $z_{i,j,n,1} = 1$, which is equivalent to $\sum_{k=1}^{K_i} z_{i,j,i,k} = 1$. Constraint (19) is equivalent with (8), observing that j is

the last intervention of technician i if and only if there exist k such that $z_{i,j,i,k} = 1$, which is equivalent to $\sum_{k=1}^{K_i} z_{i,j,i,k} = 1$.

As in the previous MILP formulation, time window constraints may induce impossible transitions. Relations (11) allows considering the same preprocessing rules than in the previous MILP formulation. A question is to analyze whether such formulation avoids sub-tours. Actually, feasible continuous sub-tours may occur, for example, $z_{i,1,2,1} = z_{i,2,1,2} = z_{i,1,2,3} = z_{i,2,i,4} = 0,5$. This configuration avoids paying a distance from 1 to 2, with a sub-tour of size 1. The following cuts can also be added to avoid such situations, similarly to previous subtour cuts $u_{i,j,j'} + u_{i,j',j} \leqslant 1 - u_{i,i,i}$.

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \forall j' \in \mathcal{J}_i, \quad \sum_{k=1}^{K_i} z_{i,j',j,k} + \sum_{k=1}^{K_i} z_{i,j,j',k} \leqslant 1 - \sum_{k=1}^{K_i} \sum_{j'' \in \mathcal{J}_i} z_{i,j'',i,k} \tag{21}$$

Note that $\sum_{k=1}^{K_i} \sum_{j'' \in \mathcal{J}_i} z_{i,j'',i,k} \in \{0,1\}$ is the flow arriving at the arrival depot. It is zero if and only if technician i has an empty route; it is the equivalent of $u_{i,i,i}$ in the first MILP formulation.

4. Matheuristics Based on Compact MILP Formulations

In this section, the previous work on MILP compact formulations is applied in order to design constructive and local search matheuristics.

4.1. MILP Neighborhoods for Local Optimizations

Neighborhoods and heuristic operators can be designed generically from an MILP formulation defining a sub MILP of reduced size (that can be parametric) and imposing values relative to some variables. Such local optimization solving sub-MILP are used in "variable fixing" heuristics (as in [5]) or in Fix-and-Optimize (FO) (as in [70]). In special cases, heuristic operators and local optimization can be implemented without using MILP computations; the computational experiments aim to provide results with respect to designing operators for the TRPTW problem and to select the promising operators that require careful implementation.

Commonly to the two last MILP formulations, different variable fixing strategies are processed on assignments $x_{i,j} \in \{0,1\}$ for $i \in \mathcal{I}$, with $x_{i,j} = 1$ if and only if job j is realized by technician i. $x_{i,j}$ is binary and a linear expression of previous variables $u_{i,j,n}$ and $z_{i,j,n',k}$.

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \quad x_{i,j} = \sum_{n \in \mathcal{N}_i} u_{i,j,n} = \sum_{k=1}^{K_i} \sum_{n' \in \mathcal{N}_i} z_{i,j,n',k} \in \{0,1\} \tag{22}$$

A constraint $x_{i,j} = 0$ for a given $i \in \mathcal{I}$ and $j \in \mathcal{J}$ imposes that variables $u_{i,j,n}, u_{i,n,j}$ or $z_{i,j,n,k}$ are zeros for all k and $n \in \mathcal{N}$. A constraint $x_{i,j} = 1$ for a given $i \in \mathcal{I}$ and $j \in \mathcal{J}$ imposes that variables $u_{i',j,n}, u_{i',n,j}$ or $z_{i',j,n,k}$ are zeros for all j', k and $i' \neq i$ with elementarity constraints (2) and (13). Such choices of fixing variables allow having balanced sub-problems as in divide-and-conquer strategies instead of the straightforward fixing on original variables $u_{i,j,n}$ and $z_{i,j,n',k}$, which results in unbalanced sub-problems. This is denoted with the fixing operator $\mathcal{F}^{assign,=}$.

$$\mathcal{F}_{i,j}^{assign,=}(b): \qquad x_{i,j} = b, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall b \in \{0,1\} \tag{23}$$

A variant is the fixing operator $F^{assign,\leqslant}$ with inequality constraints. A constraint $x_{i,j} \leqslant 0$ is equivalent to $x_{i,j} = 0$ such as previously described, and a constraint $x_{i,j} \leqslant 1$ allows another technician to realize job j.

$$\mathcal{F}_{i,j}^{assign,\leqslant}(b): \qquad x_{i,j} \leqslant b, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall b \in \{0,1\} \tag{24}$$

Previous strategies limit the assignments between jobs and technicians. One may wish to impose some orders in the interventions processed by a given technician. Additional constraints $t_j \leqslant t_{j'}$ are little nterest as they are almost always non active in the B&B search because of the weakness of big M constraints. Hence, a fixing strategy is based on the implications on the $u_{i,n,n'}$ or $z_{i,n,n',k}$ variables.

$$\mathcal{F}^{order}_{i,j,j'}: \qquad u_{i,j,j'} = \sum_{k=1}^{K_i} z_{i,j,j',k} = 0, \qquad \forall i \in \mathcal{I}, \forall (j,j') \in \mathcal{J}_i^2 \tag{25}$$

With many constraints $\mathcal{F}^{order}_{i,j,j'}$, it breaks possibilities of sub-tours, which has positive impacts for the LP relaxation quality and the B&B solving capacities of such reduced MILP.

4.2. Greedy Heuristics Iterating along Technicians

Greedy algorithms for TRPTW can be designed by iterating technician by technician. For a given technician, a local optimization can minimize the distances and the waiting times, avoiding holes that can induce scheduling a few number of jobs. Algorithm 1 implements such strategy iterating along technicians with a generic FO strategy where solutions are encoded as the assignment matrix $(x_{i,j})$.

Algorithm 1 Greedy FO heuristics iterating along technicians

Input: $\mathcal{O}$ an order to iterate among technicians, $\mathcal{F}$ a fixing strategy
Initialization: x[i][j] $= 0$
for each technician i following $\mathcal{O}$:
  define the following sub MILP of (1)–(8):
    - removing technicians after i in $\mathcal{O}$
    - and applying variable fixing strategy $\mathcal{F}$ to technicians before i in $\mathcal{O}$ with x[i][j] values
  Solve the MILP and update assignment matrix in x
end for
return assignment matrix in x and the solution cost

Variable fixing strategy in the local optimization is a crucial input parameter in Algorithm 1. Applying $\mathcal{F}^{assign,=}$ on the current solution for the previous technicians, the assignments are kept, and the optimization for the current technician i focus on the jobs that were not planned previously. In order to not seek to reoptimize a solution that was previously defined, one may completely remove this optimization and consider only technician i and the remaining jobs.

By applying $\mathcal{F}^{assign,\leqslant}$ on the current solution for the previous technicians, the current technician can take previously assigned interventions; such situations occur if it improves the distance cost and it is not possible to assign another job to the current technician that is still unplanned. To avoid reoptimization of the previous technicians, one may consider the $\mathcal{F}^{order}$ constraints to keep the previous sequence of interventions of the previous technicians and only reinserting some jobs in the planning of the current technician.

In Algorithm 1, one also defines the order to iterate among the technicians. One may iterate the following a randomized order. To take into account the multi-skilled technicians, we iterate following the increasing number of jobs among technicians; the technician can process with skill constraints so that specialized technicians can process the few jobs they have the qualification for, and multi-skilled technicians at the end of the heuristic focus on the jobs that were not previously assigned. This sort of iteration aims to maximize the chances to realize the maximal number of jobs.

4.3. Greedy Heuristics Iterating among Jobs

Another idea to process a greedy algorithm for the TRPTW is to process jobs by jobs. Algorithm 2 implements such FO strategy, where parameters include the order to iterate among jobs and an insertion operator with a maximum number of jobs for the insertion.

Algorithm 2 Greedy FO heuristic iterating along jobs

Input:
- $\mathcal{O}$ an order to iterate among jobs,
- n a maximal number of jobs for the insertion,
- `Insert`, an insertion operator defined with variable fixing strategies $\mathcal{F}$

Initialisation: x[i][j] $= 0$, `jobToPlan` = $\mathcal{J}$,
while jobs are iterated following $\mathcal{O}$:
  select $\mathcal{J}_0 \subset \mathcal{J}$, the n next jobs in `jobToPlan` following $\mathcal{O}$
  Insert optimally the jobs from $\mathcal{J}_0$ in x calling `Insert`
  remove the n first jobs in `jobToPlan` following $\mathcal{O}$
end while
return assignment matrix in x and the solution cost

Several insertion operators can be defined:

- K-insertion with order reoptimization: Inserting k chosen jobs in the current planning MILP and allowing reoptimizing of the order of jobs of a given technician are equivalent to considering the variable fixing procedure $\mathcal{F}_{i,j}^{assign,=}$ for all the jobs j in the current planning;
- K-insertion without order reoptimization: By fixing current orders, k chosen jobs can be inserted in the current planning MILP with the previous variable fixing procedure and $\mathcal{F}_{i,j,j'}^{order}$ applying for the jobs and technicians of the current planning;
- 1-insertion without order reoptimization: The previous strategy with special case $k = 1$ can be iterated easily without MILP computations. Indeed, considering the best insertion of a given job in the current planning makes at most $I + J$ possible insertions. Checking the feasibility of a sequence of job is an earliest date scheduling that applies only for improving costs in theses $I + J$ possibilities.

Another parameter concerns the sorting rules to consider jobs. The following strategies can be considered:

- Sorting the jobs by increasing the number of technicians that can process job j with skill constraints;
- Sorting the jobs by increasing value of the summed time windows where the technicians can process job j;
- Random sorting of jobs can be used to derive GRASP heuristics. We note that we can hybridize deterministic and random strategies with weighted sums to derive GRASP strategies from the deterministic order.

4.4. Local Branching Insertions of Jobs

In the previous FO strategy, the insertion order is fixed a priori. Another alternative is to consider the best insertion among the remaining jobs. Algorithm 3 implements such a strategy.

With $n = 1$, the insertion operator can try to insert each job in the current planning without modifying the order among the previously planned jobs. Enumerating all the possibilities remains polynomial. MILP optimization can be used at each iteration for any value of n, fixing the assignments out of `jobToPlan` and considering the following constraints, analogously to Local Branching [71].

$$\forall j \neq j_0, \sum_{i:\tilde{x}_{i,j}=0} (1 - x_{i,j}) + \sum_{i:\tilde{x}_{i,j}=1} x_{i,j} \leqslant n \tag{26}$$

Additional fixations for variables with MILP optimization can be the order constraints among the already planned jobs.

Algorithm 3 Best local insertion constructive heuristic

Input:
- n a maximal number of jobs for the insertion,
- $\mathcal{F}$ an optional and additional fixing strategy

Initialization: x[i][j] = 0, `jobToPlan` = $\mathcal{J}$,
for `iter` from 1 to $\left\lceil \frac{\mathcal{J}}{n} \right\rceil$:
    insert optimally n jobs of `jobToPlan` in `x`
    remove the inserted jobs from `jobToPlan`
end for
return assignment matrix in `x` and the solution cost

4.5. VND Local Search Matheuristic

Once a feasible solution is built with previous constructive matheuristics; this section improves the current solution in a local search procedure.

4.5.1. General Algorithm

We present here the general local search algorithm described in Algorithm 4, similarly to [4]. The neighborhoods are defined here using MILP definitions. MILP neighborhoods allow exploring large neighborhoods, using recent progresses of MILP solvers, having fewer and better local optimums than small neighborhoods approaches. Many MILP neighborhoods can be defined; Algorithm 4 changes systematically the choice of the neighborhood within the local search, similarly with multi neighborhood search approaches. It induces that a local optimum for the entire local search is a local optimum for all the neighborhoods considered in Algorithm 4. It ensures having less and better local optimums than classical local search approaches. The stopping criterion could be a maximal time limit or a maximal number of iterations or being in a local extremum for all neighborhoods.

Algorithm 4 VND with MILP neighborhoods

Input: an initial solutions, a set and order of neighborhoods to explore
Initialization: `currentSol` = initSolution, $\mathcal{N}$ = initial neighborhood.
while the stopping criterion is not met
    define the MILP with incumbent `currentSol` and the neighborhood $\mathcal{N}$)
    define `currentSol` as warmstart
    `currentSol` = solveMILP(MILP,timeLimit($\mathcal{N}$))
    $\mathcal{N}$ = `nextNeighborhood`($\mathcal{N}$)
end while
return CurrentSolution

Usually, MILP neighborhoods are defined for small sub-problems where exact B&B converges quickly to optimality. MILP neighborhoods are defined with three characteristics, for an efficient MILP neighborhood search includes small time limits without the optimality guarantee. Parameterization is empirical for a good trade-off between solution quality and time spent in MILP solving. The current solution is the primal solution given by the last B&B resolution, and it is also defined as warmstart for the next B&B resolution to improve the efficiency of B&B primal heuristics, enabling RINS or Local Branching heuristics from the beginning. This ensures that the solution given by the MILP resolution is at least as good as the current solution at each iteration. This algorithm is, thus, a steepest descent algorithm.

4.5.2. MILP Neighborhoods

Multiple types of large neighborhoods can be defined with the variable fixing strategies previously defined. We consider three main types of neighborhoods:

- $\mathcal{N}_i^{tec}$: For a given technician i, the neighborhood reoptimizes the routing of technician i considering any job j, even those that were planned in other routes. In other words, $\mathcal{F}_{i',j}^{assign,\leqslant}(x)$ and $\mathcal{F}_{i,j,j'}^{order}(x)$ are applied on the current solution x for all $i' \neq i$ and for all $j, j' \in \mathcal{J}$.
- $\mathcal{N}_{i,i'}^{pair}$: For a given pair of technicians i, i', the neighborhood reoptimizes only the routing of technicians i and i' considering only the jobs j planned in the current solution for i and i' and the jobs that are not assigned. In other words, the planning of the other technicians i'' is not changed, and $\mathcal{F}_{i,j}^{assign,=}(x)$ and $\mathcal{F}_{i,j,j'}^{order}(x)$ are applied on the current solution x for all $i'' \notin \{i, i'\}$ and for all $j, j' \in \mathcal{J}$.
- $\mathcal{N}_J^{jobs}$: For J a subset of jobs, planning is reoptimized fixing the assignments and the order of the jobs in $\mathcal{J} - J$. In other words, $\mathcal{F}_{i,j}^{assign,\leqslant}(x)$ and $\mathcal{F}_{i,j,j'}^{order}(x)$ are applied on the current solution x for all i and for all $j, j' \in \mathcal{J} - J$.

We note that classical neighborhoods are extended with previous MILP neighborhoods. The 2-opt and k-opt neighborhoods are included in neighborhoods $\mathcal{N}_J^{jobs}$. Insertion neighborhood, moving a job from one technician route to another, can be provided by $\mathcal{N}_i^{tec}$ or $\mathcal{N}_{i,i'}^{pair}$ neighborhoods.

4.5.3. Sequence of MILP Neighborhoods

A key point in Algorithm 4 is the sequence of neighborhoods. Applying Algorithm 4 with a single type of neighborhoods allows analyzing the impact of neighborhoods in the quality and number of local extrema. In this case, different neighborhoods are compared starting with the same initial solutions and analyzing the different qualities of the local minimum where the steepest descent local search converges. A traditional idea with VND is to increase the size of the neighborhoods when a local minimum is reached for a better compromise between computation time and the local minimum reached. Having many types of neighborhoods, the stopping criterion is firstly to reach a local minimum for all the types of neighborhoods. The choice of neighborhoods is nested, alternating deterministically the order of neighborhoods and stopping the resolution when no neighborhood can improve the current best solution.

Using neighborhoods $\mathcal{N}_i^{tec}$ or $\mathcal{N}_{i,i'}^{pair}$, the computations with the single technicians and the pairs are computed successively in a loop. After such a loop without improvement, all the classical insertions and 2-opt neighborhoods were tried so that the local minimum provided is a local minimum for all these classical neighborhoods. Using neighborhoods $\mathcal{N}_J^{jobs}$, many partitions of jobs can be designed, similarly with POPMUSIC (Partial optimization meta-heuristic under special intensification conditions, see [72]). A partition induces successive computations of $\mathcal{N}_J^{jobs}$ along the partition, and several partitions are computed successively.

4.6. Parallel Heuristics

Parallelization is a key issue for the efficiency of these heuristics. The MILP-VND local search can be naturally parallelized, exploring several (many) neighborhoods in parallel. This is particularly interesting in advanced phases of the VND when very few neighborhoods give improvements and in the last iteration in a local minimum for all the neighborhoods. The MILP-VND is a pure intensification strategy of local search, without any diversification. One of the advantage of matheuristic construction of solution is that many strategies can be designed or parametrized, as in [5]. Combining the matheuristic of this section is natural in a multi-start heuristic, starting from one solution given by one of the many constructive matheuristics and improving the solutions with MILP-VND local

search, similarly to [73]. In this case, parallelization is more useful for parallelizing the multi-start heuristic in a high level team-work parallelization.

5. Dantzig–Wolfe Reformulation and CG Matheuristics

We investigate now the extended reformulation of Danzig–Wolfe derived from the previous compact formulation and how to derive heuristics.

5.1. Extended Formulation and Column Generation

Similarly to [11], we have an extended MILP formulation by enumerating all possible (and non empty) routes $\mathcal{P}_i$ for each technician i. The cost of a route $p \in \mathcal{P}_i$ is denoted $L_{i,p}$. The variables of the MILP formulation are $z_{i,p} \in \{0,1\}$ such that $z_{i,p} = 1$ if route $p \in \mathcal{P}_i$ is chosen, and $y_j \in \{0,1\}$ with $y_j = 1$ if job j is not planned. For all $j \in \mathcal{J}$ and $p \in \mathcal{P}_i$, we used the characteristics function $\mathbb{1}_{j\in p} \in \{0,1\}$ defined with $\mathbb{1}_{j\in p} = 1$ if and only if job j is realized in route p. TRPTW can be written as the following extended MILP.

$$\min_{y,z} \quad \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{P}_i} L_{i,p} z_{i,p} + \sum_{j\in\mathcal{J}} P_j y_j \tag{27}$$

$$s.t: \quad \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p} + y_j \geqslant 1 \qquad \forall j \in \mathcal{J}, \tag{28}$$

$$\sum_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 \qquad \forall i \in \mathcal{I}, \tag{29}$$

$$z_{i,p}, y_j \in \{0,1\} \tag{30}$$

Contraints (28) express that a job j is either postponed in the case $y_j = 1$ or at least in one of the selected routes. Note that, here, elementarity is not a necessary condition; it is implied by the optimality with triangle inequality and not necessary for the validity of the MILP formulation contrary to previous compact MILP formulations. Constraints (29) enforce that at most one route is chosen for each technician, and it is possible to have only zeros for technicians that are not leaving their depots.

The difficulty is that $\mathcal{P}_i$ cannot be enumerated. The LP relaxation of this extended MILP formulation is solved by the CG algorithm, adding iteratively new routes. By having a subset of routes $\mathcal{C}_i \subset \mathcal{P}_i$, the Restricted Master Problem (RMP) denotes the previous LP relaxation applied to a subset of variables.

$$\begin{aligned} \mathrm{RMP}(\mathcal{C}_i) = \min_{z_{i,p}, y_j \geqslant 0} \quad & \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{C}_i} L_{i,p} z_{i,p} + \sum_{j\in\mathcal{J}} P_j y_j & & \\ s.t: \quad & \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{C}_i} \mathbb{1}_{j\in p} z_{i,p} + y_j \geqslant 1 & (\pi) \quad & \forall j \in \mathcal{J}, \\ & \sum_{k\in\mathcal{C}_i} z_{i,p} \leqslant 1 & (\sigma) \quad & \forall i \in \mathcal{I}, \end{aligned} \tag{31}$$

π and σ denote, respectively, the dual variables associated with the constraints (28) and (29). The RMP is always feasible as it contains a solution with empty routes for technicians and the maximal number of penalization. Using strong duality, the dual problem of continuous RMP (31) is feasible and has the same value as the primal problem.

$$\mathrm{RMP}(\mathcal{C}_i) = \max_{\pi,\sigma \geqslant 0} \quad \sum_{j\in\mathcal{J}} \pi_j - \sum_{i\in\mathcal{I}} \sigma_i \tag{32}$$

$$\forall i \in \mathcal{I}, \forall p \in \mathcal{C}_i, \quad -\sigma_i + \sum_{j\in\mathcal{J}} \mathbb{1}_{j\in p} \pi_j \leqslant L_{i,p} \quad (z_{i,p}) \tag{33}$$

$$\forall j \in \mathcal{J}, \quad \pi_j \leqslant P_j \quad (y_j) \tag{34}$$

In the dual areas, cuts have to be added until there is no violation of constraints (33). The reduced cost associated to a route (i,p) appears; $RC_{i,p} = L_{i,p} + \sigma_i - \sum_{j\in\mathcal{J}} \mathbb{1}_{j\in p} \pi_j$. Variables must be added in the RMP if their reduced cost is strictly negative. If the remaining variables have a positive reduced cost, the RMP gives the LP relaxation of (27)–(29). This

negativity question among a non-enumerable set is formulated as an optimization problem, minimizing the reduced costs of the columns that are not in $\cup_i \mathcal{C}_i$. These negativity questions are decomposed for all technicians, $RC^* = \min_i RC_i^*$, where RC_i^* is the minimization problem over the possible routes for technician i, and $RC^* \geqslant 0$ ensures that the RMP gives the LP relaxation of (27)–(29). The computations of RC_i^* are independent for the technicians and can be formulated using MILP formulations of Section 4.

$$\begin{array}{ll}
RC_i^* = & \min_u \sigma_i + \sum_{n,n' \in \mathcal{N}_i} (D_{n,n'} - \pi_j) u_{n,n'} \\
s.t: \forall j \in \mathcal{J}, & \sum_{n \in \mathcal{N}_i} u_{n,j} = \sum_{n' \in \mathcal{N}_i} u_{j,n'} \leqslant 1 \\
& \sum_{j \in \mathcal{J}_i} u_{i,j} = 1 \\
\forall j \in \mathcal{J}_i, & u_{j,j} \leqslant 0 \\
\forall j, j' \in \mathcal{J}_i, & t_j + L_j + T_{j,j'} \leqslant t_{j'} + \left(1 - u_{j,j'}\right) \cdot M_{i,j}^3 \\
j \in \mathcal{J}, & T_i^{start} + T_{i,j} \leqslant t_j + \left(1 - u_{i,j}\right) \cdot M_{i,j}^2 \\
j \in \mathcal{J}, & t_j + L_j + T_{i,j} \leqslant T_i^{end} + \left(1 - u_{j,i}\right) \cdot M_{j,j'}^1 \\
\forall j \in \mathcal{J} & t_j \in [T_j^{min}, T_j^{max}] \\
\forall n, n' \in \mathcal{N}_i & u_{n,n'} \in \{0,1\}
\end{array} \tag{35}$$

LP relaxation of (27)–(29) is calculated by Algorithm 5, with an initial set of columns as input. The initial set of columns can be empty, and RMP is always feasible thanks to the penalty costs. Matheuristics can initialize the CG algorithm adding the columns corresponding to primal solutions.

Algorithm 5 Standard column generation algorithm

Input:
$\mathcal{C}$ set of initial columns.
do:
solve RMP (31) with columns defined in $\mathcal{C}$
store dual variables σ and π and optimal cost from (31)
for each technician $i \in \mathcal{I}$:
solve (35) to optimality with last (σ, π) values
if $CR_i^* < 0$ **then** add the optimal column to $\mathcal{C}$
end for
while: columns are added in $\mathcal{C}$
return the last cost of the RMP (31)

Remark 1. *Constraints (28) are written with inequalities, and the MILP model is still valid with equalities instead. One could also have equalities in (29), defining explicitly empty routes. Using inequalities induces a better stability of the column generation algorithm, with signed dual variables π, σ [11].*

5.2. POPMUSIC-CG Decomposition

Dual variables associated with jobs can be interpreted as marginal costs to realize such jobs. When continuous RMP does not use any variable visiting a job j, we have $\pi_j = P_j$. When the dual variables are not stabilized, it induces sub-problems generating columns with jobs with the highest values π_j in (35). Hence, independent computations of (35) are likely to visit the same jobs, with the highest values of π_j. These generated columns are likely to be redundant in the recombination induced by the next RMP computation. It suggests incorporating diversification in the CG algorithm. We investigate combined sub-problems for a subset of technicians $\mathcal{I}_0 \subset \mathcal{I}$ (typically two or three technicians), imposing a total diversification in these sub-problems with the following MILP formulation.

$$
\begin{array}{ll}
RC^*_{\mathcal{I}_0} = & \min_{u,t} \sum_{i \in \mathcal{I}_0} \left(\sigma_i + \sum_{n,n' \in \mathcal{N}_i} (D_{n,n'} - \pi_j) u_{i,n,n'} \right) \\
s.t : \forall j \in \mathcal{J} & \sum_{i \in \mathcal{I}_0} \sum_{n \in \mathcal{N}_i} u_{i,j,n} \leqslant 1 \\
\forall i \in \mathcal{I}_0, \forall j \in \mathcal{J}, & \sum_{n \in \mathcal{N}_i} u_{i,n,j} = \sum_{n' \in \mathcal{N}_i} u_{i,j,n'} \\
\forall i \in \mathcal{I}_0, \forall n \in \mathcal{N}_i, & \sum_{n' \in \mathcal{N}_i} u_{i,n,n'} = 1 \\
\forall i \in \mathcal{I}_0, \forall j \in \mathcal{J}, & u_{i,j,j} \leqslant 0 \\
\forall i \in \mathcal{I}_0, \forall (j,j') \in \mathcal{J}^2, & t_j + L_j + T_{j,j'} \leqslant t_{j'} + \left(1 - u_{i,j,j'}\right) \cdot M^3_{i,j} \\
\forall i \in \mathcal{I}_0, \forall j \in \mathcal{J}, & T^{start}_i + T_{i,j} \leqslant t_j + \left(1 - u_{i,i,j}\right) \cdot M^2_{i,j} \\
\forall i \in \mathcal{I}_0, \forall j \in \mathcal{J}, & t_j + L_j + T_{i,j} \leqslant T^{end}_i + \left(1 - u_{i,j,i}\right) \cdot M^1_{j,j'} \\
\forall j \in \mathcal{J} & t_j \in [T^{min}_j, T^{max}_j] \\
\forall i \in \mathcal{I}_0, \forall n \in \mathcal{N}_i, \forall n' \in \mathcal{N}_i, & u_{i,n,n'} \in \{0,1\}
\end{array}
\tag{36}
$$

Sup-problem (36) can be seen as a concatenation of the technician sub-problems (35) in $\mathcal{I}_0 \subset \mathcal{I}$. The first constraint enforces that each job can be affected with respect to at least one technician in $\mathcal{I}_0$ for a total diversity among the new columns. One may consider a multi-objective extension and generate columns for $\mathcal{I}_0 \subset \mathcal{I}$ and consider the partial diversity in another objective.

The process gives rise to Algorithm 6, where the partitions of technicians vary each iterations, similarly to POPMUSIC heuristics (Partial optimization metaheuristic under special intensification conditions [72]). Note than POPMUSIC matheuristics are efficient as primal heuristics without computations of dual bounds for vehicle routing problems [74].

Algorithm 6 POPMUSIC column generation algorithm

Input:
  $\mathcal{C}$ set of initial columns.
do:
  solve RMP (31) with columns defined in $\mathcal{C}$
  store dual variables σ and π and optimal cost from (31)
  compute $\mathcal{P}_I$ a partition of $\mathcal{I}$ in small subsets
  for each subset $\mathcal{I}_0 \in \mathcal{P}_I$:
    solve (36) with a matheuristic with last (σ, π) values
    for each column c with a negative reduced cost
      add the column to $\mathcal{C}$
  end for
  end for
while: columns are added in $\mathcal{C}$
return the last cost of the RMP (31) and updated set of columns $\mathcal{C}$

5.3. Solving CG Sub-Problems

To solve sub-problems (35) with respect to optimality, note that the CG sub-problem related to i is exactly the same as the VRPTW sub-problems with the jobs $\mathcal{J}_i$. Constraint (35) can be formulated as an elementary shortest path problem with resource constraints (ESPPRC). If ESPPRC is a NP-complete problem, it is solved efficiently for reasonable instance sizes for VRP problems with labeling algorithms [23,40]. With several technicians in sub-problems (36), the nature of sub-problem changes and exact labeling algorithms will be even more difficult. In this study, generic MILP solvers are used to solve exactly such sub-problems and to validate the contribution of such a decomposition scheme.

However, the computations to optimality are required only to prove the optimality of the RMP at the last iteration. For the first CG iterations, we only need to generate negative reduced cost solutions. The exact computations are time consuming in Algorithm 5; it can be replaced by heuristics to generate quickly negative reduced cost solutions. The choice of algorithm at each iteration seeks for the good trade-off between computation time and quality (negativity) of the generated columns. This trade-off is dealt using different heuristics with different computation times. For instance, the heuristics of Section 4 apply. In Algorithm 2, specific sorts of jobs can be processed for CG sub-problems: One may consider firstly the jobs with the highest dual values π_j, especially with the non-assigned jobs in the RMP with a maximal dual value $\pi_j = P_j$, rather than the jobs with a low dual value. Hence, one may iterate in Algorithm 2 the decreasing values of π_j. One may also consider $\pi_j - D_{i,j}$ in order to take into account the remoteness of job j.

Sub-problems (36) can be solved to search columns with a negative reduced cost using matheuristics; the number of technician is restricted, and matheuristics restrict the size of MILP computations, removing jobs similarly to Section 4. The POPMUSIC CG scheme applies only for generating columns with a negative reduced cost. To prove the optimality of the RMP, Algorithm 5 applies for the remaining iterations.

Algorithm 7, decomposes the search of columns with a negative reduced cost by using only computations with one technician. These decomposed sub-problems can also be solved as ESPPRC with [23]. This scheme allows generating the best individual columns as in Algorithm 5 and also good complementary columns for these best individual columns. The solutions of (35) with Algorithm 7 are uneven in the reduced costs following the order of priority of the technicians, whereas a straightforward matheuristic search applied to (35) would have a smoother repartition of the reduced costs. A stake of the computational experiments is to determine whether these two generation schemes are complementary or if one dominates the other. To generate both types of columns, Algorithm 7 can be applied to generate initial columns before a smoothing local search phase in order to optimize the objective function of (36) with a MILP-VND similar to Section 4.

Algorithm 7 Diversification of sub-problem solutions

Input:
- $\mathcal{I}_0$ a subset of technician.
- s a cyclic permutation of $\mathcal{I}_0$ with order$(s) = \mathcal{I}_0$.
- σ, π the dual variables of the last RMP computation.

Initialization: $\mathcal{C} = \varnothing$, the columns to add in the RMP
for each technician $i \in \mathcal{I}_0$:
  Let $i' := i$, $\mathcal{J}_0 := \mathcal{J}$
  for $k = 1$ to $\mathcal{I}_0$:
    solve (35) for technician i' with (σ, π) values and the remaining jobs in $\mathcal{J}_0$
    if the solution induces a column with a negative reduced cost
      add the column in $\mathcal{C}$
      remove the jobs of the column in $\mathcal{J}_0$
    $i' := s(i_0)$
  end for :
end for
return $\mathcal{C}$

5.4. How to Select Partitions of Sub-Problems?

A key point for the developments of Section 5.2 and Algorithm 6 is the choice of the partition of sub-problems. Having a subset of technicians with disjoint competences, the joint sub-problem (36) is decomposable, an optimal solution is obtained to concatenate optimal solutions of single sub-problems (35) and there is no benefit to consider grouped sub-problem (36) instead of the standard ones. This highlights the importance of selecting technicians with the most similar skills. Note that technicians differ with the locations of the depots; this point does not allow having exactly symmetrical sub-problems that can be

considered as one sub-problem type with a multiplicity of columns that can be used in the RMP, as in [12].

5.4.1. Formulation as Clustering Problems

A first approach is to define a distance among technicians and a null distance for technicians having exactly the same number of skills so that clustering technicians induces minimizing intra-cluster distances. A Hamming distance can be used in that goal. A first approach would be to consider the Hamming distance of the skill subsets. In order to consider the most similar sub-problems (35) in clusters, it is actually better to use the Hamming distance in the vector of J Booleans, indicating the possibility of a technician to realize a given job. With our notation, this distance d among technicians can be written as follows.

$$\forall i, i' \in \mathcal{I}, \quad d_{i,i'} = \sum_{j \in \mathcal{J}} \left(\mathbb{1}_{j \in \mathcal{J}_i} \times \mathbb{1}_{j \notin \mathcal{J}_{i'}} + \mathbb{1}_{j \notin \mathcal{J}_i} \times \mathbb{1}_{j \in \mathcal{J}_{i'}} \right) \tag{37}$$

A second point before using the clustering algorithm is to define the number of clusters. Here, the crucial point in the application is to deal with a restricted number of technicians in each sub-problems, which is relatively small because of the difficulty of solving sub-problems (36), typically $N = 2$ or 3 in our numerical experiments. Thus, we consider $P = \left\lceil \frac{I}{N} \right\rceil$ clusters, and we consider a clustering problem such as P-means or P-medoids with a cardinality constraint that each cluster is of size at most N.

Having cluster of sizes $N = 2$ or 3 brings about the fact that cardinality constrained adaptations of standard local search such as Lloyd's heuristic (also called k-means algorithm) are not adapted frameworks. One can use simple greedy heuristics to create such clusters. In Algorithm 6, it is also useful to diversify the partitions. Several good partitions can be constructed by using a randomized greedy algorithm, for instance, using one of the three best possible insertions in a randomized manner instead of choosing the best local insertion.

5.4.2. MILP Formulation with Two Technician by Cluster

In the case where technicians are grouped by at most two, the clustering optimization can be formulated with an MILP formulation similar to assignment problems. By defining binary variables $x_{i,i'} \in \{0,1\}$ for all $i < i'$ $x_{i,i'} = 1$ if and only if technicians i, i' are grouped in the same cluster, the clustering optimization can be formulated as following MILP.

$$\min_{x_{i,i'}} \quad \sum_{i<i'} d_{i,i'} x_{i,i'} \tag{38}$$

$$\sum_{i'>i} x_{i,i'} + \sum_{i'<i} x_{i',i} \leqslant 1 \qquad \forall i \in \mathcal{I} \tag{39}$$

$$\sum_{i \in \mathcal{I}} \left(1 - \sum_{i'>i} x_{i,i'} - \sum_{i'<i} x_{i',i} \right) \leqslant 1 \tag{40}$$

$$x_{i,i'} \in \{0,1\} \qquad \forall i < i' \in \mathcal{I} \tag{41}$$

Constraints (39) enforce that each technician is associated to at most one cluster so that clusters are of size at most two. Constraint (40) enforces that at most one technician is not related to another one. Note that such situation is unavoidable and happens when the number of technician is odd. In the case of even numbers of technician, one can remove constraint (40) and consider an equality in (39). In this case, the problem is very similar with assignment problems, which ensures that the MILP formulation is efficient for large values of $\mathcal{I}$. Even in the general case, the small values render solving ILP easy. One can also consider an extension with at most three technicians by cluster; the bottleneck is the resolution of Constraint (36) here and not optimal computations of such partitions.

As previously mentioned, it is useful in Algorithm 6 to have diversified partitions. By solving the previous MILP formulation, it is easy to have one optimal solution, and many solutions with similar costs can be generated using local moves with 2-opt exchanges. Indeed, considering only competences, there exist many symmetries with 2-opt exchanges with technicians having the same skill sets.

5.4.3. Dealing with Heterogeneity of Sub-Problems

Sub-problems can be very asymmetric in terms of sizes, dealing with different subset of jobs $\mathcal{J}_i$ with the skill constraints. At the end of the CG algorithms, the sub-problems dealing with less jobs are more likely to have finished the CG process with fewer columns to generate. Some sub-problems can be skipped a priori for some iterations. More generally, CG sub-problem solving should be more frequent for sub-problems with the higher cardinality of $\mathcal{J}_i$. Deterministically, the frequency of solving a sub-problem i can be set to $\mathcal{J}_i/\mathcal{J}$. Another criterion is to focus on the sub-problems that produced the most negative columns in the last iterations. One can also use a Reinforcement Learning algorithm like ε-greedy approach to update the probability of choosing a sub-problem. For such approach, rewards model that columns with a negative reduced cost are found, and one may also count separately as another reward that columns with a significatively negative reduced cost are found.

5.4.4. Related ML Techniques for Decomposition in Mathematical Programming

We mention here the related ideas using ML techniques for decomposition methods in mathematical programming. Learning techniques can also be useful for a CG algorithm to solve sub-problems using the optimal solutions of previous sub-problems, as in [17]. In this work, it allows speeding up the resolution for sub-problems, which is the most time consuming part in CG algorithms. Such ideas are not implemented in our approach; it is complementary with the ML-guided stabilization proposed in this paper. We note also similarities with [9] in the case of a Bender's decomposition structure rather than a DW decomposition. In [9], standard sub-problems are also grouped, and an appropriate clustering problem is defined and solved with ILP formulations an matheuristics.

5.5. Tabu Search Intensification

A general difficulty to implement CG algorithms is that dual variables converge erratically. This is closely related to properties of linear optimization as the optimums are extreme points. A consequence for CG algorithm is that many iterations are necessary to have good dual variables to generate the most appropriate columns, with an erratic convergence of dual variables. Stabilization techniques enforce smoothing the convergence of dual variables in order to reduce the number of iterations that converge for the CG algorithm and, thus, reduce the computation time needed for CG convergence. Mathematical properties can not only be exploited for stabilization but also heuristics [75].

This section is motivated by the following fact: Many good columns can be obtained with slight modifications from a good column. At one iteration, high values of π indicate the jobs that are likely used in a new column. A kernel of jobs with high π_j values can be combined in several interesting columns regarding the reduced costs. Using such a property, it is interesting to aggressively generate columns with local search moves from interesting columns. This section introduces a Tabu Search (TS) matheuristic to generate quickly a pool of good solutions.

Let $x_{i,j} = \sum_{n' \in \mathcal{N}_i} u_{i,j,n'}$, the binaries indicating if technician $i \in \mathcal{I}_0$ realizes job j. Having M feasible solutions as previously calculated, we denote with $\tilde{x}^m_{i,j}$ the value of these variables. To forbid already generated columns, we add the following "no-good-cuts" constraints similarly to [76], which defines a variable fixing strategy.

$$\forall m \in [\![1, M]\!], \sum_{i,j:\tilde{x}^m_{i,j}=1} (1 - x^m_{i,j}) + \sum_{i,j:\tilde{x}^m_{i,j}=0} x^m_{i,j} \geqslant 1 \tag{42}$$

In order to search around the last solution $\tilde{x}^M_{i,j}$, allowing K modifications from the M-th solution, it can also be written as a linear constraint that defines a variable fixing strategy.

$$\sum_{i,j:\tilde{x}^M_{i,j}=1} (1 - x^M_{i,j}) + \sum_{i,j:\tilde{x}^M_{i,j}=0} x^M_{i,j} \leqslant K \tag{43}$$

These constraints allow generating a TS procedure in Algorithm 8 in order to aggressively generate columns. In the case of the classic CG algorithm, the input for $\mathcal{I}_0$ is a singleton. Adding such constraints in the CG sub-problems can be solved with Branch and Bound for small MILP computations. For larger MILP computations, these constraints respect the structures of the MILP matheuristics in order to quickly find good solutions. We note that the aggressive generation of negatively reduced cost columns with a tabu search was also introduced in [77] for a significative acceleration of CG in practice.

Algorithm 8 Tabu search intensification

Input:
- $\mathcal{I}_0$ a subset of technicians
- the current value in the RMP of dual variables (σ, π)
- a set of initial columns $c \in \prod_{i \in \mathcal{I}_0} \mathcal{P}_i$ with a negative reduced cost in (35)
- an integer $M \in \mathbb{N}$, a maximal number of TS iterations
- an integer $K \in \mathbb{N}$, a maximal number of assignment modifications

TSINTENSIFICATION$(\mathcal{I}_0, K, M)$
//Initialization:
MILP, a MILP formulation for (35) related to technician i
$p = c$ initial columns
Taboo list of columns $l = \{c\}$
an integer $n = 0$ to denote iterations
//Loop to generate columns with negative reduced costs
do:
Add constraint (43) in MILP with columns of p
Add constraint (42) in MILP with columns of p
solve MILP
remove constraint (43) in MILP with columns of p
update p, the optimal columns in the last MIP
update l adding the columns of p with a negative reduced cost in l
while $n < M$ and ReducedCost$(p) < 0$
return l // the list of columns to add in the next RMP

5.6. Dual Bounds Derived from CG

As mentioned before, when Algorithms 5 or 7 are terminated with the non existence of columns with negative reduced costs, the value of the continuous RMP is a lower bound of the problem, which is the LP relaxation of the problem (27)–(29). There is a theoretical advantage in computing the LP relaxation of the extended formulation (27)–(29) instead of one of the LP relaxation of the compact formulations [78]. However, it is possible to have equality among these LP relaxations in the case of a tight formulation existing for sub-problems, as in [79]. It is a numerical issue to test if an improvement of the LP relaxation is obtained considering the DW extended formulation and if it compensates the higher computation times to process the CG algorithm.

Computing the LP relaxation of the extended formulation requires at least solving each sub-problems to optimality with non strictly negative optimal values. Actually, the conditions to derive the lower bounds are weaker. At each iteration of the CG algorithm, one can compute dual bounds by using an equivalence between DW decomposition and Lagrangian relaxation.

$$\begin{array}{lll} v = \min\limits_{y,z} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} P_j y_j & \\ s.t: & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p} + y_j \geqslant 1 & \forall j \in \mathcal{J}, \\ & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & z_{i,p}, y_j \in \{0,1\} & \end{array} \tag{44}$$

It can be reformulated as a min-max problem.

$$\begin{array}{lll} v = \min\limits_{y,z}\max\limits_{\lambda} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} P_j y_j + \sum\limits_j \lambda_j(1 - y_j - \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & z_{i,p}, y_j \in \{0,1\}, \lambda_j \geqslant 0 & \end{array}$$

$$\begin{array}{lll} v = \min\limits_{y,z}\max\limits_{\lambda} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} (P_j - \lambda_j) y_j + \sum\limits_{j\in\mathcal{J}} \lambda_j(1 - \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & z_{i,p}, y_j \in \{0,1\}, \lambda_j \geqslant 0 & \end{array}$$

For optimality, it is necessary that $\lambda_j \leqslant P_j$. Thus, we have the following.

$$\begin{array}{lll} v = \min\limits_{y,z}\max\limits_{\lambda} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} (P_j - \lambda_j) y_j + \sum\limits_{j\in\mathcal{J}} \lambda_j(1 - \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & \lambda_j \leqslant P_j & \forall j \in \mathcal{J}, \\ & z_{i,p}, y_j \in \{0,1\}, \lambda_j \geqslant 0 & \end{array}$$

Swapping minimization and maximization provides a lower bound.

$$\begin{array}{lll} v \geqslant \max\limits_{\lambda}\min\limits_{y,z} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} (P_j - \lambda_j) y_j + \sum\limits_{j\in\mathcal{J}} \lambda_j(1 - \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & \lambda_j \leqslant P_j & \forall j \in \mathcal{J}, \\ & z_{i,p}, y_j \in \{0,1\}, \lambda_j \geqslant 0 & \end{array}$$

The independent optimization in y induces that $y_j = 0$. Thus, we have the following.

$$\begin{array}{lll} v \geqslant \max\limits_{\lambda}\min\limits_{z} & \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} L_{i,p}z_{i,p} + \sum\limits_{j\in\mathcal{J}} \lambda_j(1 - \sum\limits_{i\in\mathcal{I}}\sum\limits_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum\limits_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & 0 \leqslant \lambda_j \leqslant P_j & \forall j \in \mathcal{J}, \\ & z_{i,p} \in \{0,1\} & \forall i \in \mathcal{I}, \forall p \in \mathcal{P}_i \end{array}$$

In other words, we have following families of dual bounds:

$$v \geqslant l(\lambda), \quad \forall \lambda \in \prod_{j\in\mathcal{J}} [0, P_j]$$

where $l(\lambda)$ is a Lagrangian bound defined by the following.

$$\begin{array}{lll} l(\pi) = \sum_j \lambda_j + \min_z & \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{P}_i} L_{i,p} z_{i,p} + \sum_{j\in\mathcal{J}} \lambda_j (1 - \sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{P}_i} \mathbb{1}_{j\in p} z_{i,p}) & \\ s.t: & \sum_{p\in\mathcal{P}_i} z_{i,p} \leqslant 1 & \forall i \in \mathcal{I}, \\ & 0 \leqslant \lambda_j \leqslant P_j & \forall j \in \mathcal{J}, \\ & z_{i,p} \in \{0,1\} & \forall i \in \mathcal{I}, \forall p \in \mathcal{P}_i \end{array}$$

The reduced cost at each iteration of CG algorithm with dual values σ_i, π_j appears with $\lambda_j = \pi_j$. The reduced cost minimization $RC_i^* = \sigma_i + \min_{z_{i,p}, p\in\mathcal{P}_i}(L_{i,p} - \sum_{j\in k} \pi_j) z_{i,p}$ appears, and we have the following.

$$l(\lambda_j) = \sum_{j\in\mathcal{J}} \pi_j - \sum_{i\in\mathcal{I}} \sigma_i + \sum_{i\in\mathcal{I}} RC_i^* \tag{45}$$

Relation (45) makes a relation between the Lagrangian bound $l(\lambda)$ related to the dual signal (π, σ) and the optimal values of the sub-problems RC_i^*. Using only heuristic resolution of sub-problems to reach stabilized value of the dual signal (π, σ), optimal computations of each sub-problem RC_i^* allow having a lower bound, which is a Lagrangian bound. We note that having only dual bounds for all RC_i^*, with some LP relaxations or dual bounds given by a truncated B&B search allows computing the lower bounds of the Lagrangian bound (45), which is possible even for large size instances where optimal computations of sub-problems RC_i^* are not possible. Note that if the CG algorithm terminates, the dual bounds is $\sum_{j\in\mathcal{J}} \pi_j - \sum_{i\in\mathcal{I}} \sigma_i$; the Lagrangian bounds the value of the continuous RMP, and this result was also obtained by duality in (32).

5.7. Primal CG Heuristics

Once the LP relaxation (31) is computed by CG algorithm, the question is how to repair LP relaxation into integer solutions. The following strategies can be processed:

- RMP_INT_HEUR(): Solving the integer RMP with the generated columns is a first quick heuristic. Such a heuristic was used in [33,46];
- LAG_HEUR(): The continuous assignments $\tilde{x}_{i,j} = \sum_p \mathbb{1}_{j\in p} \tilde{z}_{i,p} \in [0,1]$ implied by continuous relaxation of RMP $\tilde{z}_{i,p}$ can be repaired to obtain primal solutions, similarly with Lagrangian heuristics (the equivalence between CG iterations and Lagrangian relaxation is shown in previous section). Variable fixing strategies, fixing assignments with values 0 and/or 1, can be a first strategy. Once variables are fixed, straightforward B&B search or specific heuristics may apply in order to quickly obtain solutions;
- LAG_RINS(): RMP_INT_HEUR() and LAG_HEUR() can be combined in LAG_RINS(), a RINS lagrangian heuristic fixing common values in the continuous assignments $\tilde{v}_{i,j}$ and in the assignments of the primal solution given by RMP_INT_HEUR(). With such an approach, the initial solution from RMP_INT_HEUR() is still feasible for the RINS optimization for a better warmstarting of the Branch& Bound search or heuristics setting this initial solution.

We note firstly that these repairing heuristics can be processed at each iteration of the CG scheme, which would induce using such repair heuristics in a multi-start environment provided by the new columns and LP relaxations at each iteration. We note secondly that these heuristics can also be used in a B&P tree.

5.8. Management of the CG Pool

In the standard CG Algorithm, one column is generated by each sub-problem resolution. When many iterations are processed, it may be useful to pay attention to the number of columns in the RMP. Having a very large number of columns may cause memory errors or prohibitive computation times to solve RMP problems. In practice, the column pool

may be cleaned regularly, for example, by defining a threshold in the number of columns to remove after the next RMP computation. Nonbasic columns, i.e., with a null value in the RMP computation, may be removed. It is possible that the removed columns are re-generated later; the benefit of the removal is in the reduction in column pool and its implications. Several criteria can determine which columns to remove, for example, storing the last used column in a RMP computation and removing the columns that were used the less recently or computing for all the columns the reduced cost with the updated value of dual variables and removing the columns with the highest reduced cost [78]. It is advised that the individual should avoid using this procedure too often and to keep a certain number of nonbasic columns in the master problem after column removal [78].

With the developments of this paper, especially with Algorithm 8, the columns may be generated very aggressively. This induces calling the column removing procedure more often. A specific strategy for the column pool management is proposed here, taking into account that some columns are useful also in the integer RMP computations to define an integer solution, even if the column is not interesting for the continuous RMP computations and, thus, the convergence of the CG algorithm. In our management strategy of the column pool, the following rules are provided. The procedure is enabled only when the number of columns reach a given threshold. The procedure is called every $C \in \{3, 5\}$ iterations of the CG scheme. The columns used in the C computations of the continuous RMP are marked and kept for the following. A computation of the integer RMP is processed after these C RMP iterations in order to mark and keep the best primal solution (and possibly removing jobs that are used at least twice as in the post processing procedure of Section 5.6). The remaining columns are deleted following the updated value of the reduced cost with the last computation of dual variables in order to have a constant and defined number of columns to reiterate CG iterations.

5.9. Using Parallel Computing and a Many-Core Environment

Although the CG algorithm follows a sequential structure adding iteratively new columns, several points may be parallelized to speed up the computations in practice. Firstly, the computations of sub-problems are independent for each technician. The $\mathcal{I}$ independent computations of sub-problems shall be parallelized. The parallelization to speed up the phases of the sub-problems resolution is the main issue for the speed up, as the sequential part with the LP computations of RMP (31) is less time consuming. In order to optimize the load balancing of the parallel implementation, a LPT (Lowest Processing Time) strategy is natural, using increasing preprocessing times which are predicted with the sizes of the sub-problems with skill constraints. Furthermore, the taboo search intensification proposed in Algorithm 8 allows a more fine-grain parallelization. Lastly, parallelization can be also used when dealing concurrently with several heuristic strategies to generate columns with a negative reduced cost and to perform several of the many variants of heuristics. Having many possibilities of hybridization allows taking profit from a many-core environment.

6. Computational Results

Numerical experiments were computed with a quadcore computer with Intel(R) Core(TM) i7-4790S CPU 3.20 GHz with 8 threads and running Linux Ubuntu, with 16 Gb of RAM memory.

6.1. Instance Characteristics, Benchmarks and Analysis Methodology

6.1.1. Instance Characteristics

Instances were generated for this study; these instances are available online: https://github.com/ndupin/SiteDepTRPTW (accessed on 25 October 2021). Instances are named gotic_a_b_c_ex$d$.txt where $a \in \{3, 5, 8, 10, 15, 20\}$ is the number of technicians, $b \in \{1, 3, 5, 10, 20\}$ is the number of skill types, $c \in \{10, 20, 40, 50, 80, 100\}$ is the number of jobs to plan and $d \in [\![1, 8]\!] \cup [\![11, 18]\!]$distinguishes instances generated with the same

previous characteristics. For each instance, jobs have the same penalization cost. Penalties are larger than transportation costs so that the optimal solutions serve a maximal number of jobs. For only instance gotic_15_20_40_ex7.txt, it was impossible to realize all the jobs; the other optimal solutions visit every customer. In the instances, localisation of depots and customers is given with 2D coordinates, and the distance from localisation coordinates (x, y) and (x', y') shall be computed using the following rule (different from [19]).

$$d\big((x,y),(x',y')\big) = \left\lfloor \sqrt{\lfloor (x-x')^2 \rfloor + \lfloor (y-y')^2 \rfloor} \right\rfloor \tag{46}$$

The (symmetric) commuting time between locations (x, y) and (x', y') is given by the following:

$$t\big((x,y),(x',y')\big) = \left\lfloor \frac{d((x,y),(x',y'))}{speed} \right\rfloor \tag{47}$$

where *speed* is a single speed parameter defined in the instances.

6.1.2. Analysis Methodology

Instances were generated with relations among instances to analyze the trade-off between optimality and feasibility for the different optimization algorithms. Instances with $d \in [\![11, 18]\!]$ are generated from instances with $d - 10 \in [\![1, 8]\!]$ with a higher speed parameter. Hence, each feasible solution for an instance with $d \in [\![1, 8]\!]$ is feasible for the corresponding instance with parameter $10 + d$,

For each group of instances with fixed values of parameters a and b defined, a maximal number of competences are defined with c_{max}. The other instances with $c < c_{max}$ are defined from the one with c_{max}: skills in the instance s are grouped modulo c; if skill q is required in instance c_{max}, the corresponding skill in instance c is q mod c. Such reduction keeps the feasible solutions from the previous model and adds potentially new solutions. Hence, instances with $c < c_{max}$ are less constrained than the original ones with $c = c_{max}$ and have a better optimal value. In the case $c = 1$, there are no skill constraints, and the model is similar with a MDVRPTW. For computational analyses, we investigate the difficulty of instances regarding speed and competence parameters. This defines four datasets:

- Dataset sc_{max} with minimal speed ($d \in [\![1, 8]\!]$) and maximal number of competences c_{max};
- Dataset Sc_{max} with maximal speed ($d \in [\![11, 18]\!]$) and maximal number of competences c_{max};
- Dataset $s1$ with minimal speed ($d \in [\![1, 8]\!]$) and no skill constraint ($c = 1$);
- Dataset $S1$ with maximal speed ($d \in [\![11, 18]\!]$) and no skill constraint ($c = 1$).

Hence, datasets $s1$ and $S1$ are MDVRPTW instances, with a correspondence and more constrained instances due to speed with dataset $s1$. Dataset $S1$ is less constrained than the three other datasets. Original dataset sc_{max} is the more constrained dataset, comparable to the three other datasets. Comparing the difficulty induced for instances sc_{max} and $S1$ allows analyzing the difficulty induced by speed and rare competence separately.

In order to compare optimization algorithms in a given dataset, the following indicators are provided:

- Fail: The number of instances where the maximal number of jobs is not scheduled;
- BKS: The number of instances where the Best Known Solution (BKS) is reached;
- Mean: The average gap for all the considered instances of the primal solution to the BKS;
- σ: The standard deviation of the gaps between primal solution and BKS;
- Min: The minimum gaps between primal solution and BKS for all instances ;
- Q1: The first quartile of the gaps between primal solutions and BKS;
- Med: The median (or second quartile) of the gaps between primal solutions and BKS;
- Q3: The third quartile of the gaps between primal solutions and BKS;
- Max: The maximum gaps between primal solution and BKS for all instances.

6.1.3. Implementation and Characteristics of MILP Solving

The numerical experiments reported here used Cplex version 12.6 to solve MILP and LP. The analysis of B&B characteristics was provided by using the OPL modeling language to call Cplex. The matheuristics of Section 4 were implemented with OPL script to iterate successive MILP computations. A first CG scheme was implemented with OPL script also by using the populate mode to observe the impact of keeping several columns with negative reduced cost per sub-problem resolution. The final CG scheme and heuristics for sub-problems were coded in Python by using the PuLP library to call for MILP and LP. Using PuLP allows using Cplex or open source tools (such COIN-OR) the same programming interface. However, using PuLP resulted in important computation time to load MILP models, similarly to https://python-mip.readthedocs.io/en/latest/bench.html (accessed on 25 October 2021). PuLP is not a relevant choice to solve iteratively many MILP models in short computation times, and the loading time of the MILP model may be much longer. Hence, the computation times were not representative with such implementations. The results of the study conclude the types of heuristics that can be used, which include hybridized and parallelized heuristics for future and better implementations.

6.1.4. LocalSolver

As a benchmark local search solver for our study, we used LocalSolver version 7.5. We note that the results are better now by using the last versions of LocalSolver; we provide here the results with version 7.5, and our instances were given to the LocalSolver Company for their internal benchmark. For the latest versions of LocalSolver (at least with version 10), BKSs are found for most of the instances within 600 s solving time. Note that our instances may be interesting for benchmarking general local search solvers, especially to test the accuracy of local search facing highly constrained instances where some multi-start strategies may be useful. Benchmark codes with LocalSolver are also given in https://github.com/ndupin/SiteDepTRPTW (accessed on 25 October 2021). For the efficiency of the LocalSolver model, we used the definition of variables using LocalSolver lists, and TW constraints are modeled as an objective function accumulating TW infeasibilities, with a hierarchical optimization considering firstly feasibility and then the costs of the routes. Such a choice is similar with LocalSolver's VRPTW example and was validated by LocalSolver experts. Two variants were implemented. The general case considers only the disjoint constraint among Localsolver's lists: the routes of each technician visit disjoints sets of jobs. For the special case where a solution exits without outsourcing costs, one can model it with a partition by adding a cardinality constraint with LocalSolver. The reported results show the results with the general variant; failures are cases where a non-maximum number of jobs are assigned to technicians.

6.1.5. Solving MILP Compact Formulations

Analyzing the characteristics of the B&B straightforward resolution for the different variants of MILP formulations of Section 3 is a key issue for determining how to choose the MILP formulation for the next developments. Table 2 compares the quality of the dual bounds for the MILP formulations (1)–(8), (12)–(19) and (27)–(29) on instances solvable to optimality with a straighforward B&B resolution. For the compact formulations (1)–(8) and (12)–(19), the bounds are compared with and without the cuts (10) or (21) for the LP relaxation and the dual bound at the root node of the B&B tree before branching. For the extended formulation, only the LP relaxation is computed with the CG algorithm. This table highlights that the quality of the linear relaxation of the compact formulation is very weak, with a large gap relative to the Lagrangian bounds. The cuts of Cplex provide a significant improvement of the LP relaxation in all the cases. The cuts (10) or (21) provide also significant improvements that are not redundant with the cuts for Cplex. The FGG-based formulation with the cuts (21) gives significantly the best dual bounds among the compact formulations.

Table 2. Comparison of the average gaps to dual bounds given by some variants of MILP formulations to the optimal solutions on a set of 55 instances solvable to optimality with Cplex. The results are gathered on instances with the same number of technicians, skills and interventions. LP denotes the linear relaxation, whereas nod1 denotes the bounds obtained after the cutting plane passes of Cplex at the root node before branching.

Bound **Cuts**	**(1)–(8)** **LP** **No**	**(1)–(8)** **nod1** **No**	**(1)–(8)** **LP** (10)	**(1)–(8)** **nod1** **(10)**	**(12)–(19)** **LP** **No**	**(12)–(19)** **nod1** **No**	**(12)–(19)** **LP** **(21)**	**(12)–(19)** **nod1** **(21)**	**(27)–(29)** **LP** **No**
data_3_1_10	45.7%	8.7%	11.9%	3.2%	28.1%	3.3%	8.8%	1.0%	1.8%
data_3_5_10	43.4%	0.0%	9.7%	0.0%	23.4%	0.0%	8.0%	0.0%	0.0%
data_5_1_20	44.8%	29.5%	27.9%	11.6%	35.7%	17.9%	22.0%	13.7%	1.9%
data_5_3_20	52.7%	30.3%	30.8%	17.7%	39.0%	19.9%	22.7%	15.5%	0.6%
data_5_5_20	58.1%	28.7%	27.5%	14.6%	42.1%	18.5%	25.1%	14.1%	0.3%
data_8_1_20	40.5%	20.6%	20.8%	10.6%	32.6%	10.6%	16.8%	7.8%	1.3%
data_8_3_20	54.7%	28.5%	28.6%	17.1%	35.2%	14.8%	20.0%	11.2%	1.8%
data_8_5_20	60.6%	26.7%	25.9%	11.2%	39.7%	8.1%	21.3%	1.6%	0.5%
data_10_10_40	55.9%	25.0%	25.8%	14.3%	40.8%	19.4%	20.7%	13.9%	2.0%
data_15_10_40	55.6%	18.9%	19.6%	10.7%	35.8%	11.0%	13.4%	6.3%	0.6%
data_15_20_40	53.9%	14.2%	16.1%	6.1%	32.1%	7.5%	13.1%	4.8%	0.7%

6.2. MILP Formulations Results

By choosing the MILP formulations and the cuts to add for the integer resolution, these results on the dual bounds shall be compared with the computation time. Indeed, adding cuts (10) or (21) for choosing the FGG-based formulation increases the size of the MIP, which induces more computation time for computing LP relaxations. Let us analyze the best compromise in the improvement of LP relaxation relative to the time spent to compute the dual bound. Table 3 compares the computation time for solving small problems to optimality with the previous variants of compact MILP formulations. The FGG-based formulation has significantly worse results than formulation (1)–(8) with the cuts (10). Adding the cuts (10) or (21) is justified also in the computations to optimality by Table 3.

An explanation of the huge gap between the lightest and the FGG-based MILP formulation comes with the efficiency of the generic primal heuristics. For the different compact formulations, the generic primal heuristics provided bad results to find first solutions, and convergence of primal bound is very long. It is indeed difficult to find solutions with the maximal number of jobs allocated in the first stages of the resolution even for medium size instances. Giving a good initial solution as warmstart to solvers allowed improving significantly the computation time for solving instances to optimality, cutting off earlier a significant number of nodes in the B&B tree. These facts are related to the poor quality of the LP relaxation. The empirical results showed that the efficiencies of primal heuristics were significantly worse with the FGG-based MILP formulations, which explains the difference observed in Table 3.

6.2.1. Solving CG Sub-Problems

If CG sub-problems with one technician having a similar polyhedral structure with the entire compact MILP formulation, the efficiency of straightforward MILP solving is very different. For some mid size instances, it can be less time consuming to solve the entire MILP with a compact formulation rather than solving a single technician CG sub-problem. It is paradoxical, and DW reformulation decomposes the original MILP with iterations of smaller problems. The reason for such paradoxical situations is that sub-problems are of a different nature. Seeing CG sub-problems as shortest path problems, many distances are negative in providing a negatively reduced cost. In the original MILP problems, all the distances are positive. For sub-problems, cycles with negatively reduced cost may appear, and this tends to produce many sub-tours with negatively reduced costs in continuous

relaxations. When solving sub-problems with many sub-tours of negative costs with an MILP solver, the lower bounds are of a poor quality and many sub-tours are generated in many nodes of the B&B tree. This induces that the convergence to optimality is difficult in this case.

Table 3. Comparison of computation time (in seconds) of both compact MILP formulations of Section 3, with or without the 2-sub-tour cuts.

	(1)–(8) No Cuts	(1)–(8) +(10)	(12)–(19) No Cuts	(12)–(19) +(21)
4_1_20_ex4	34.7	7.6	333.1	601.2
4_3_20_ex4	45.4	19.6	799.3	121.2
4_5_20_ex4	14.6	5.4	35.0	27.0
5_3_20_ex1	131.5	83.4	>1200	955.5
5_3_20_ex2	21.3	24.9	224.8	86.2
5_3_20_ex3	34.7	14.4	>1200	647.3
5_3_20_ex4	48.1	20.4	259.9	223.8
5_5_20_ex1	4.3	5.4	161.6	36.1
5_5_20_ex2	5.5	2.6	34.0	3.0
5_5_20_ex3	6.2	5.8	118.4	101.9
5_5_20_ex4	7.5	4.1	40.4	29.2
8_5_20_ex1	8.4	6.1	81.4	12.5
8_5_20_ex2	1.0	0.7	0.4	0.4
8_5_20_ex3	2.5	0.8	1.7	1.9
8_5_20_ex4	5.3	0.9	0.8	1.8
8_5_20_ex5	2.1	1.0	5.7	3.6
15_20_40_ex1	46.2	7.5	408.7	89.1
15_20_40_ex2	8.3	3.0	22.4	15.0
15_20_40_ex3	15.6	10.6	53.9	31.2
15_20_40_ex4	58.8	15.7	108.3	35.6
15_20_40_ex5	114.5	15.2	114.4	58.6
15_20_40_ex6	6.9	3.7	4.7	5.2
15_20_40_ex8	7.3	1.5	3.3	2.7

Note that this critical situation appears more often in the first iterations of the CG algorithm when the dual values are not stabilized and have values of π_j; in the worst case $\pi_j = P_j$ occurs when penalization is used in the RMP in the case of unbalanced dual variables. In the last iterations of the CG algorithm, having more stable dual variables induces less complications, and straightforward MILP solving is more efficient at these iterations compared to the first iterations. A good point is that primal heuristics of MILP solvers (relying on branching, in particular, diving heuristic) remains efficient for finding columns with a negative reduced cost for sub-problems in the first iterations, which is enough to continue the CG algorithm without solving optimality the first sub-problems. Note that for a given CG iteration, the computation efficiency of sub-problems i significantly varies following the number of jobs that the technician considered in a sub-problem can realize. At the end of the CG algorithm, it tends to solve quickly the sub-problems with the less jobs, proving that no column of negative reduced cost exist, and the sub-problems with remaining columns to add are the sub-problems with the maximal number of jobs with the competence constraint.

With the different nature of MILP optimization searching for columns with negative reduced costs, the differences between the FGG formulation and the most compact MILP formulation are emphasized, with larger gaps in the quality of LP relaxations. In order to provide dual bounds of sub-problems, as in Section 5.7, it is better to use the FGG formulation. For using the primal heuristics and solving in truncated time in matheuristics, the most compact formulation (1)–(8) is more efficient. Note that the efficiency of the sub tour cuts is lessened (10).

6.2.2. Results with Extended DW Reformulation

Table 2 shows that the quality of the lower bounds of the extended DW reformulation outperforms compact formulations at the LP relaxation and also at the root noot of the B&B tree considering the generic cuts implemented in Cplex. Note that this was not the case in [79], and these results are coherent with the known results for the class of vehicle routing problems [11].

For many instances, the LP relaxation of the extended formulation has no gap with the integer solution, and the integer RMP already provides an optimal integer solution. For such instances, no branching is needed for a B&P enumeration scheme. Note that such results occurs more often for the highly constrained instances. For lowly constrained instances, continuous combinations of columns often exist for decreasing the RMP objective function. Skill constraints induce some decoupling of technician routes and are helpful for CG approaches. Combined with TW constraints and a limited number of jobs per route, it is in favor of CG approaches with the generation of few and diversified columns.

6.3. Results with Constructive Matheuristics

Table 4 compares the efficiency of the constructive matheuristics over all instances. These results can be compared to the other approaches over the same set of instances in Table 5; it illustrates that these heuristics are not efficient in terms of accuracy in finding feasible solutions and in terms of the quality of the feasible solutions found. Generally, reoptimizing previous decisions involves $\mathcal{F}^{assign,\leqslant}$ instead of $\mathcal{F}^{assign,=}$, such as standard greedy approaches. Contrary to [5], considering in parallel these different constructive matheuristics and taking the best solution found induces significant over costs for most of the instances, and it is not enough to find a feasible solution for all the instances.

6.3.1. Constructive Matheuristics Iterating Technician by Technician

Constructive heuristics of Section 4.2 iterating technician by technician are particularly inefficient, and many infeasibilities and mostly solutions of a poor quality result. The local optimization tends to produce dense plannings to assign a maximal number of jobs in the first iterations and last iterations with few jobs to assign. This shape of solution explains the poor results of these heuristics. Firstly, the optimal solution may be very different from this shape of solutions with unbalanced solutions, especially for the easiest instances. Even the reoptimization with $\mathcal{F}^{assign,\leqslant}$, it is not efficient to balance the routes of technicians; a reason for this is that the first routes have also been optimized with distances. Thus, removing jobs induces few impacts. Such variable fixing does not allow placing another job in the previously computed routes. Secondly, the local optimizations do not take into account that some jobs have to be assigned in the planning for rarity reasons; it explains the poor accuracy of these heuristics. These results conclude that the construction of solution "technician by technician" shall be avoided in a specific implementation.

6.3.2. Constructive Matheuristics Iterating by Buckets of Jobs

The constructive heuristics of Section 4.3 iterating with buckets of jobs are more efficient. The results of Table 4 illustrate that slight improvements are obtained using $\mathcal{F}^{assign,\leqslant}$ rather than $\mathcal{F}^{assign,=}$ and that considering the TW amplitude to possibly realize jobs is better than considering only the number of technicians able to realize the job. The reason is that rarity of jobs is a key element in the order of local optimizations. Increasing the number of jobs in the local optimizations tends to increase the quality of solutions, and some counter-examples are found where the 10 jobs optimization results in worse results than using the five jobs optimization. Indeed, even if the 10 jobs optimization contains twice the amount as five jobs optimization and would be seen as better local optimization, the optimality of local optimization in greedy algorithms is not a guarantee of the quality of the final solution. Local Branching insertions of Section 4.4 provides aggressive heuristics that are able to find much better solutions but also frequently results

in solutions that do not allocate the maximal number of jobs, especially for the most constrained instances.

Table 4. Comparison of the statistical indicators presented in Section 6.1.2 using several constructive matheuristics for all the instances. *n*job denotes Algorithm 2 grouping in subsets of n jobs, and 1tic denotes Algorithm 1. For job insertion, the initial sorting of jobs is based on the number of technicians that can realize the jobs (nbTec) or with the cumulated time considering TW constraints (timeJob). Variable fixing in MILP local optimization includes $\mathcal{F}^{assign,\leqslant}$ or $\mathcal{F}^{assign,\leqslant}$.

	Sort	Fix	Fail	BKS	Mean	σ	min	Q1	Med	Q3	Max
1 job	nbTec	$\mathcal{F}^{assign,=}$	7	0	21.54%	11.49%	5.48%	12.83%	20.72%	28.36%	44.57%
2 job	nbTec	$\mathcal{F}^{assign,=}$	7	0	20.01%	10.07%	5.79%	11.94%	19.34%	26.02%	40.35%
8 job	nbTec	$\mathcal{F}^{assign,=}$	5	0	16.59%	8.27%	2.74%	10.90%	16.15%	21.26%	34.56%
10 job	nbTec	$\mathcal{F}^{assign,=}$	6	0	16.27%	8.55%	3.89%	10.80%	14.89%	19.51%	36.64%
1 job	timeJob	$\mathcal{F}^{assign,=}$	2	0	28.61%	15.62%	7.91%	17.65%	26.14%	36.95%	66.30%
2 job	timeJob	$\mathcal{F}^{assign,=}$	2	0	26.04%	13.95%	7.24%	17.02%	23.44%	32.77%	61.49%
5 job	timeJob	$\mathcal{F}^{assign,=}$	2	0	23.68%	12.12%	5.10%	15.19%	21.39%	32.85%	47.61%
8 job	timeJob	$\mathcal{F}^{assign,=}$	2	0	19.94%	11.23%	3.03%	11.32%	19.00%	26.81%	44.76%
10 job	timeJob	$\mathcal{F}^{assign,=}$	1	0	19.74%	11.37%	5.61%	11.95%	16.47%	24.33%	51.71%
1 job	nbTec	$\mathcal{F}^{assign,\leqslant}$	6	0	19.58%	9.78%	4.68%	12.57%	18.94%	26.45%	37.34%
2 job	nbTec	$\mathcal{F}^{assign,\leqslant}$	6	0	18.68%	9.43%	5.64%	11.61%	17.29%	24.25%	39.60%
5 job	nbTec	$\mathcal{F}^{assign,\leqslant}$	3	0	6.18%	2.80%	1.36%	4.24%	5.57%	8.48%	11.40%
8 job	nbTec	$\mathcal{F}^{assign,\leqslant}$	7	0	20.83%	11.35%	3.63%	13.83%	20.07%	26.22%	48.52%
10 job	nbTec	$\mathcal{F}^{assign,\leqslant}$	6	0	4.69%	1.33%	1.90%	4.32%	4.87%	5.55%	6.65%
1 job	timeJob	$\mathcal{F}^{assign,\leqslant}$	2	0	27.12%	15.91%	6.69%	16.09%	24.29%	35.62%	64.39%
2 job	timeJob	$\mathcal{F}^{assign,\leqslant}$	2	0	24.41%	13.12%	6.31%	15.40%	21.89%	31.12%	57.20%
5 job	timeJob	$\mathcal{F}^{assign,\leqslant}$	4	0	21.85%	11.40%	4.87%	13.97%	18.85%	29.41%	44.54%
8 job	timeJob	$\mathcal{F}^{assign,\leqslant}$	4	0	18.04%	10.34%	2.96%	10.73%	17.21%	23.74%	40.87%
10 job	timeJob	$\mathcal{F}^{assign,\leqslant}$	6	0	18.34%	10.74%	5.30%	11.10%	15.80%	22.09%	45.33%
1 tic		$\mathcal{F}^{assign,=}$	10	0	55.31%	27.98%	24.83%	36.67%	48.13%	66.72%	123.43%
1 tic		$\mathcal{F}^{assign,\leqslant}$	10	0	54.32%	25.52%	25.59%	35.88%	49.45%	65.79%	116.51%
Best	Constr		1	0	13.17%	8.99%	1.84%	7.57%	11.97%	17.20%	38.70%

Table 5. Comparison of the solution quality of local search approaches for the complete dataset. LocalSolver results are reported for defined time limits, CG heuristics and VND matheuristics at termination. For VND matheuristics, different initializations are provided, and the multi-start VND considers the reported initialization for a parallel or multi-start implementation.

	Fail	Mean	σ	min	Q1	Med	Q3	Max
LocalSolver 30 s	2	6.05%	4.20%	0.85%	2.73%	5.06%	8.12%	16.04%
LocalSolver 60 s	2	4.50%	3.51%	0.64%	2.11%	3.58%	6.38%	14.41%
LocalSolver 300 s	2	1.96%	1.96%	0.00%	0.45%	1.29%	2.69%	7.14%
LocalSolver 600 s	2	1.03%	1.01%	0.00%	0.14%	0.90%	1.57%	3.51%
LocalSolver 900 s	1	0.94%	1.06%	0.00%	0.03%	0.40%	1.57%	3.51%
RMP_INT_HEUR	0	1.07%	1.27%	0.00%	0.05%	0.63%	1.60%	3.90%
LAG_HEUR	0	0.73%	1.01%	0.00%	0.01%	0.34%	1.06%	3.46%
LAG_RINS	0	0.20%	0.34%	0.00%	0.00%	0.04%	0.25%	1.18%
LAG_RINS+VND	0	0.10%	0.26%	0.00%	0.00%	0.00%	0.06%	0.97%
5 job,nbTec+VND	0	1.90%	2.29%	0.00%	0.23%	0.92%	2.80%	7.31%
5 job,timeJob+VND	0	2.03%	2.34%	0.00%	0.25%	1.09%	3.16%	8.38%
10 job,nbTec+VND	0	2.38%	2.67%	0.00%	0.51%	1.57%	3.47%	10.38%
10 job,timeJob+VND	0	2.93%	3.00%	0.02%	0.77%	2.19%	4.28%	10.84%
init0+VND	0	3.45%	3.76%	0.00%	0.82%	2.49%	5.09%	13.82%
Multi-start VND	0	0.70%	1.10%	0.00%	0.03%	0.19%	0.86%	9.23%

6.3.3. CG Heuristics

As previously mentioned, the LP relaxation of the extended formulation has no gap with the integer solution for many instances, especially highly constrained instances. The integer RMP already gives an optimal integer solution; thus, the post-processing heuristics furnish an optimal solution. Table 5 shows the average results for all the larger instances, and the partial results on the subsets for the comparative analyses are provided in Table 6. In the cases where the integer RMP is not optimal, the solution quality is still excellent and much better than the constructive matheuristics based on the compact formulations. Significant improvements are provided, removing jobs that are realized twice or more in the integer RMP solution, such situation appeared multiple times. Significant and quick improvements are then provided with RINS diving using compact formulation, providing high quality solutions with a majority of BKS.

The difficulty with CG heuristics is the time required to reach the convergence of the CG algorithm. Compact matheuristics are much faster. Speeding up the convergence of the CG exact algorithm is a crucial issue, and analyses are reported in a following section. Using only heuristics and matheuristics to solve sub-problems allows having quick heuristics. However, it induces a significant decrease in solution quality for continuous and integer RMP. Using only heuristics without matheuristic operators for solving sub-problems almost keep some jobs not realized even in the continuous RMP. After such quick phase with heuristic solving of sub-problems, using matheuristics (including the matheuristic stabilization operators) allows accurately finding feasible solutions for all of the instances. These two phases are worthwhile for reaching the quickest computation time such as the stages of the CG convergence compared to the use of exact resolutions of sub-problems. To derive the CG heuristics, some few iterations of the CG algorithm are required with exact solutions sub-problems with the stabilization operators. Solving sub-problems with LocalSolver in a short time limit and applying TS matheuristic intensification with Algorithm 8 was also very efficient in that context.

6.4. Results with MILP-VND Local Search

Tables 5 and 6 provide the comparative results among local search approaches. For the MILP-VND matheuristics, it allows comparing the quality of the local minimum obtained by using different initialization heuristics. The comparative analyses on subsets of instances in Table 6 illustrate the impact of the difficulty of constraints for the efficiency of local search approaches.

6.5. Stabilization of CG Algorithm

If the extended formulation and the CG algorithm allows providing lower and upper bounds of an excellent quality, the critical issue is to make the CG algorithm converge quickly, which is the bottleneck of this method. As previously outlined, using quick heuristics in the first phase and then matheuristics in the second phase with the TS intensification as matheuristic operator allows quicker iteration than using an exact method, and it is the quickest way to reach advances phases of the CG algorithm, but it is not enough to reach the convergence of the CG algorithm. Note that this is particularly interesting for avoiding the exact methods at the first iterations with respect to solving the most difficult sub-problems, as described in Section 6.2.1.

This section provides the results applied to decreasing the number of iterations of the CG algorithm in terms of converging faster in order to validate the different CG schemes. Two experiences with exact methods are provided in this section, and the validation of CG schemes has also an impact for designing the first iterations of the CG algorithm with heuristic and matheuristic resolution of CG sub-problems.

Table 6. Comparison of the solution quality of local search approaches for the 17 mid-size instances of dataset sc_{max}, Sc_{max} and S1. LocalSolver results are reported for defined time limits, CG heuristics and VND matheuristics at termination. For VND matheuristics, different initialization are provided, and the multi-start VND considers the reported initialization for a parallel or multi-start implementation.

For Dataset sc_{max} (the Most Constrained Instances)									
	Fail	**BKS**	**Mean**	σ	**min**	**Q1**	**Med**	**Q3**	**Max**
LocalSolver 30 s	2	0	13.68%	9.93%	2.43%	6.47%	10.78%	19.09%	35.86%
LocalSolver 60 s	2	0	11.24%	9.61%	2.31%	3.76%	6.99%	19.03%	32.18%
LocalSolver 300 s	2	0	5.36%	5.78%	1.30%	1.96%	3.35%	6.96%	24.52%
LocalSolver 600 s	2	0	3.89%	2.63%	0.70%	1.91%	3.35%	5.06%	8.76%
LocalSolver 900 s	1	1	2.99%	2.37%	0.00%	1.68%	2.49%	2.95%	8.26%
RMP_INT_HEUR	0	6	1.09%	1.32%	0.00%	0.00%	0.50%	1.82%	3.66%
LAG_HEUR	0	7	0.81%	1.10%	0.00%	0.00%	0.43%	1.07%	3.66%
LAG_RINS	0	8	0.19%	0.29%	0.00%	0.00%	0.08%	0.29%	1.12%
LAG_RINS+VND	0	15	0.08%	0.27%	0.00%	0.00%	0.00%	0.00%	1.12%
5 job,nbTec+VND	0	3	2.91%	2.84%	0.00%	0.49%	2.10%	4.71%	9.23%
5 job,timeJob+VND	0	2	3.60%	3.59%	0.00%	0.70%	2.66%	5.56%	14.32%
10 job,nbTec+VND	0	1	3.34%	3.17%	0.00%	1.24%	2.84%	4.43%	13.53%
10 job,timeJob+VND	0	0	4.71%	3.57%	0.09%	2.33%	4.13%	7.03%	11.67%
init0+VND	0	2	5.91%	6.04%	0.00%	1.11%	4.44%	10.46%	20.91%
Multi-start VND	0	4	1.71%	2.33%	0.00%	0.11%	0.66%	2.49%	9.23%
For dataset sc_{max}									
	Fail	**BKS**	**Mean**	σ	**min**	**Q1**	**Med**	**Q3**	**Max**
LocalSolver 30 s	0	0	7.91%	5.36%	1.32%	4.97%	7.05%	8.56%	24.33%
LocalSolver 60 s	0	0	5.02%	3.06%	0.79%	2.94%	4.80%	6.55%	13.50%
LocalSolver 300 s	0	3	1.85%	1.63%	0.00%	0.56%	1.30%	2.98%	4.62%
LocalSolver 600 s	0	5	1.20%	1.45%	0.00%	0.03%	0.77%	1.77%	4.62%
LocalSolver 900 s	0	5	0.91%	1.09%	0.00%	0.03%	0.51%	1.24%	3.63%
RMP_INT_HEUR	0	5	1.31%	1.78%	0.00%	0.03%	0.48%	2.03%	5.52%
LAG_HEUR	0	7	0.87%	1.45%	0.00%	0.00%	0.13%	1.02%	4.76%
LAG_RINS	0	9	0.14%	0.18%	0.00%	0.00%	0.06%	0.25%	0.59%
LAG_RINS+VND	0	16	0.03%	0.08%	0.00%	0.00%	0.00%	0.00%	0.27%
5 job,nbTec+VND	0	7	1.29%	1.99%	0.00%	0.00%	0.40%	1.78%	7.42%
5 job,timeJob+VND	0	7	1.29%	1.99%	0.00%	0.00%	0.40%	1.78%	7.42%
10 job,nbTec+VND	0	8	1.80%	3.17%	0.00%	0.00%	0.17%	2.24%	12.66%
init0+VND	0	7	0.77%	1.03%	0.00%	0.00%	0.22%	0.95%	3.16%
Multi-start VND	0	12	0.29%	0.55%	0.00%	0.00%	0.00%	0.33%	2.06%
For Dataset S1 (the Less Constrained Instances)									
	Fail	**BKS**	**Mean**	σ	**min**	**Q1**	**Med**	**Q3**	**Max**
LocalSolver 30 s	0	0	7.78%	5.03%	2.56%	3.44%	6.65%	11.21%	19.30%
LocalSolver 60 s	0	0	6.35%	4.67%	0.83%	2.97%	5.24%	7.46%	15.81%
LocalSolver 300 s	0	3	2.42%	3.05%	0.00%	0.13%	0.69%	4.62%	9.10%
LocalSolver 600 s	0	6	1.71%	2.59%	0.00%	0.00%	0.32%	1.56%	7.52%
LocalSolver 900 s	0	8	0.83%	1.19%	0.00%	0.00%	0.13%	1.46%	3.72%
RMP_INT_HEUR	0	3	0.86%	0.80%	0.00%	0.16%	0.79%	1.29%	2.21%
LAG_HEUR	0	5	0.35%	0.41%	0.00%	0.02%	0.17%	0.67%	1.36%
LAG_RINS	0	11	0.17%	0.31%	0.00%	0.00%	0.00%	0.13%	0.86%
LAG_RINS+VND	0	16	0.07%	0.21%	0.00%	0.00%	0.00%	0.00%	0.86%
5 job,nbTec+VND	0	9	0.67%	0.91%	0.00%	0.00%	0.05%	0.89%	2.53%
5 job,timeJob+VND	0	9	0.67%	0.91%	0.00%	0.00%	0.05%	0.89%	2.53%
10 job,nbTec+VND	0	8	0.99%	1.21%	0.00%	0.00%	0.47%	1.87%	3.86%
init0+VND	0	11	0.68%	1.51%	0.00%	0.00%	0.00%	0.72%	5.98%
Multi-start VND	0	15	0.07%	0.19%	0.00%	0.00%	0.00%	0.00%	0.74%

Using only complementary neighborhoods with pairs of technicians $\mathcal{N}_{i,i'}^{pair}$ and reoptimization of buckets of 10 jobs $\mathcal{N}^{jobs}$ in different partitions allowed considering larger neighborhoods than the traditional ones, contains the other main neighborhood and allowed quick resolution times. As in [4], a very short number of iterations of the MILP-VND

using iteratively all the considered neighborhoods were enough for converging to a local minimum; for most instances, 2/3 of such iterations were needed, and the worst cases used five such iterations. MILP-VND allows having quick convergence and scaled better for all the larger instances than exact methods. Contrary to [4], such large neighborhoods are not sufficient to converge to local minimums of excellent quality. The quality of the local minimum varies significantly following the different initialization. Considering the four matheuristic initialization and the null initialization, followed by the MILP-VND provided a multi-start matheuristic that is accurate for finding good solutions, the most difficult dataset sc_{max} provided the worst solutions. Dataset sc_{max} is generally the more difficult one for local search approaches, with less feasible neighbor moves because of the hard constraints. This was not the case of the CG-based heuristics. Using MILP-VND after the LAG_RINS heuristic of an excellent quality allowed polishing the results and increases the number of BKS found, illustrating the advantages, drawbacks and complementarity of local search approaches and CG heuristics depending on the structures of the instances.

6.5.1. Impact of CG Initialization and Multi-Column Generation per Sub-Problem

In this section, we analyze the results considering the standard Algorithm 5 with an exact resolution of sub-problems and generating one or several columns per sub-problem. The second variant is implemented by using the populate mode of Cplex with OPL; we note that it is possible to use all the columns with a negative reduced cost found using callbacks, which is not available with OPL but is now available in some MILP modeling interfaces and libraries such as PythonMIP or Julia JUMP. The generation of several columns is used at most for the five first iterations, and the remaining iterations are processed with the standard Algorithm 5 with at most one column generated per sub-problem. The maximal size of the population of Cplex is set to ten: at most, 10 new columns are added in the RMP.

Figure 1 illustrates the convergence of integer and continuous RMP in both variants; the results for some small instances are given with the number of iterations to converge in Table 7. Figure 1 illustrates that CG is naturally stable for TRPTW, with a few degenerated iterations. Degenerescence occurs more frequently for VRPTW [11]. A reason is that the I sub-problems are highly heterogeneous with difference competence sets and location of depots; a degenerated iteration would induce the generation of I degenerate columns, whereas in VRPTW, only one sub-problem is generated when taking into account the symmetry among vehicles, and the probability of degenerate iteration is larger [11]. Figure 1 shows that integer and continuous RMP are very close; it happens for highly constrained instances and also with generation of few columns. It explains the high quality of the RMP_INT_HEUR heuristic.

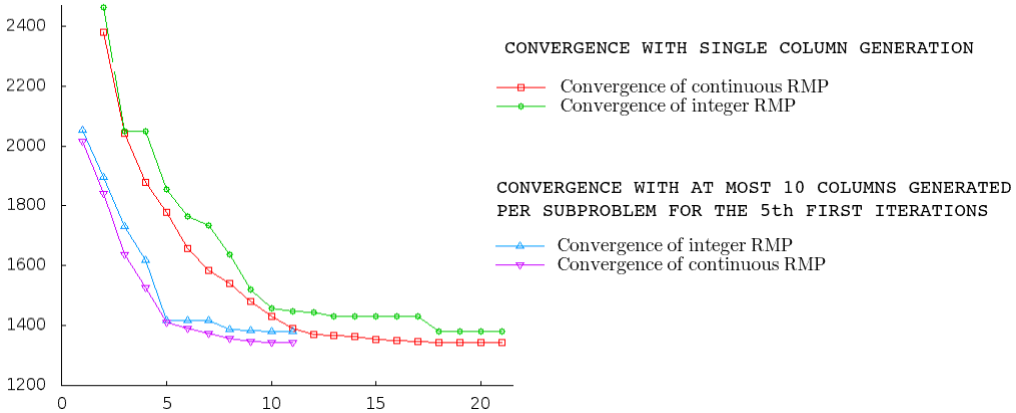


Figure 1. Illustration of the CG convergence of RMP and integer RMP for an instance with 40 jobs and 15 technicians with 20 different skills.

Using several solution found by the MILP solver induces many benefits for the CG convergence: The difference in the quality of continuous and integer RMP is significant, and the dual variables have a less erratic convergence. This is a virtuous circle: Having better RMP values induce better dual values that generate more relevant columns. This explains how the experiments with only five iterations of aggressive CG produce such difference. The first iterations are the most unstable.

Table 7 considers, furthermore, an initialization with the optimal integer solution. In many cases, the convergence value of the continuous RMP is obtained here, and there are only degenerated iterations with the optimal iterations. In such cases, the dual initial values are very good, but it is not sufficient to guarantee the fast convergence of the CG Algorithm 5. If the optimal initialization makes the number of iterations decrease on average, there exists some instances where there are more degenerate iterations with the optimal initialization than iterations starting from no column.

Table 7. Number of iterations for the convergence of CG algorithm to solve the LP relaxation of the extended formulation compared when only one column is generated per sub-problem, improvements with multi-column generation and providing an optimum of the MILP as the initial solution.

Instance	1 col per Sub-Problem	Multicol	Multicol + Warm Start
data_3_5_10_ex1	24	7	4
data_3_5_10_ex2	13	5	7
data_3_5_10_ex3	8	5	5
data_3_5_10_ex4	11	6	5
data_8_5_20_ex1	21	9	7
data_8_5_20_ex2	11	9	7
data_8_5_20_ex3	38	15	12
data_8_5_20_ex4	15	10	8
data_15_20_40_ex1	21	13	13
data_15_20_40_ex2	19	11	9
data_15_20_40_ex3	17	10	14
data_15_20_40_ex4	21	11	11
data_10_3_50_ex1	54	21	22
data_15_3_50_ex1	32	26	18
Total	324	168	150
% iterations	100	51.9%	46.3%

These experiments validate the impact of having a multi-column generation even with an unstructured method of generating columns. The next section provides the results related to the structured method of generating several columns per sub-problem.

6.5.2. Impact of the Different CG Schemes and Stabilization Operators

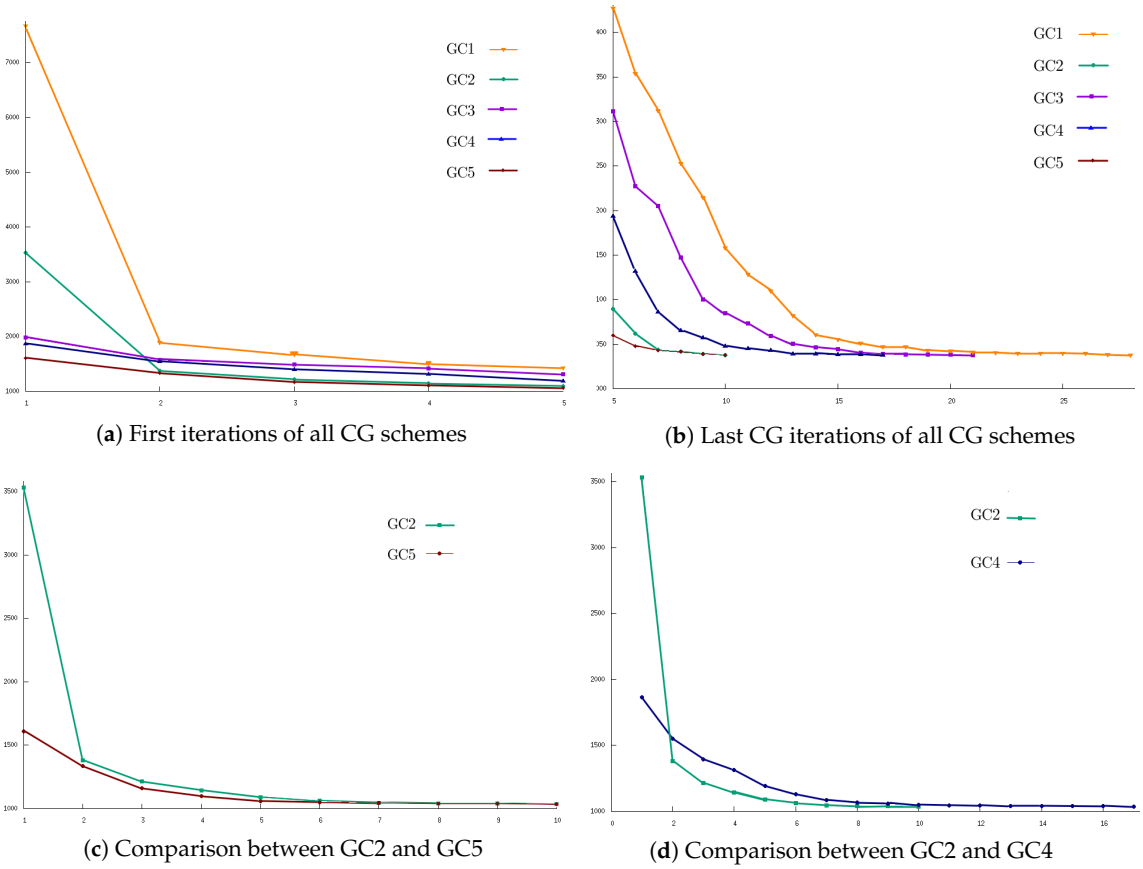
To understand the specific contributions of the different CG schemes , we provide here comparative analyses solving exactly sub-problems for small and mid-size instances and a proven termination of the CG algorithm. The following CG schemes are implemented and compared:

- `CG1`: This is the standard CG scheme, as written in Algorithm 5;
- `CG2`: The standard CG scheme is followed by TS intensification to solve single technician sub-problems. It corresponds to Algorithm 8 with only partitions into singletons, with parameters $k = 3$ and $N = 5$;
- `CG3`: This is the POPMUSIC CG scheme without TS intensification, where sub-problems (36) are solved only using exact computations;
- `CG4`: This is the POPMUSIC CG scheme without TS intensification, where the sub-problems (36) are solved twice: Algorithm 7 provides the first solutions in a hierarchical manner, and a second phase optimizes the worst reduced cost (and, thus, balances

the reduced costs) with a VND local search matheuristic, generating all the columns with a negatively reduced cost obtained from these two phases.

- `CG5`: This corresponds to the strategy `CG4` with TS intensification activated with $K = 3$ and $M = 5$.

The dominant computational results are illustrated in Figure 2. The known hierarchies among CG schemes are significant: `CG1` is significantly dominated by the other schemes, POPMUSIC and TS intensification induce a very significant acceleration in terms of CG convergence. Comparing `CG3` and `CG4` allows concluding that the optimization of the total reduced cost with diversification in sub-problems (36) induces a better CG convergence than the CG with the hierarchical decomposition of Algorithm 7. Combining POPMUSIC and TS intensification dominates the single stabilization strategies: POPMUSIC column generation is mostly useful for the first iterations in terms of recombining columns in the next RMP, where it can be difficult to find a solution allocating all the jobs (and, thus, paying no prohibitive penalty). TS intensification is crucial when dual variables have a relative stability, which is helpful for the latest phases of the CG convergence. When the dual variables are changing slightly, it is likely that there exist many columns of very good quality among neighbors of the optimal solution of the previous sub-problem. Aggressively using TS intensification prevents generating neighbors from the previous iterations in further iterations, which explains the gain when using TS intensification. To compute the extended LP relaxation to optimality, TS strategies are prominent for reducing the number of CG iterations. TS is particularly efficient for the final iterations of CG algorithm, but it is also useful in the first iterations with respect to having a deterministic scheme to aggressively generate solutions in the first iterations, as in the first experience of stabilization.



(**a**) First iterations of all CG schemes

(**b**) Last CG iterations of all CG schemes

(**c**) Comparison between GC2 and GC5

(**d**) Comparison between GC2 and GC4

Figure 2. Comparison of the CG convergence for an instance with 40 jobs and 15 technicians with 10 different skills. For a better readability, all CG schemes are separately illustrated on the very first and the termination iterations, and comparisons between GC2 and GC5 or GC4 are shown in separate graphs.

7. Conclusions and Perspectives

This work provided several results and offers different perspectives. The CG algorithm is efficient for computing dual and primal bounds for the TRPTW coherently with the results obtained for VRPTW problems. The LP relaxation of the DW extended formulation efficiently guides us toward primal solutions of an excellent quality.Matheuristics based on the compact ILP formulation may be inefficient even with combinations of large neighborhoods in a VND local search, contrary to [4], or with parallelization using a portfolio of matheuristics, contrary to [5]. We note also a similarity with [25]: improving the exact methods and MILP formulations is a first step before applying MILP models to matheuristics, and the first step is crucial for the efficiency of the resulting matheuristic.

If CG heuristics are particularly efficient in terms of quality, the key point is to speed up and stabilize the convergence of the CG algorithm. Two matheuristic components are proposed and proven efficient for TRPTW. On the one hand, the heterogeneity of sub-problems allows a POPMUSIC decomposition to generate columns by avoiding redundancy in the served requests for a better re-combination in the RMP computation. ML techniques are used to generate appropriate partitions, illustrating the impact of hybridizing ML and ILP techniques. On the other hand, a Tabu matheuristic intensification for sub-problems avoids generating similar columns in neighbor iterations of the CG algorithm, decreasing the number of iterations to converge. POPMUSIC matheuristic is efficient for the first CG iterations where it is difficult to serve all the requests, whereas TS intensification remains efficient for the last phases of the CG algorithm where dual values are quite stable. For both schemes, heuristic solving schemes are designed to speed up the search of negative reduced cost columns, and light heuristics are firstly processed and then heavier matheuristics are used speed up the time of first CG iterations. A perspective would be to improve the exact solutions of such sub-problems for the validated CG scheme by using labelling algorithms. Note that TS intensification and many matheuristics are compatible with labelling algorithms; a perspective is to implement such matheuristics without using an MILP solver. Possessing many strategies and hybridizations for generating columns is a good characteristic with parallelization in a multi-core environment.

We note that our final approach investigates and combines the three types of matheuristics for VRPTW problems in the taxonomy of [15]: *CG-based heuristics* as a general framework, with computation of primal and dual bounds; *improvement heuristics* with local search procedures to improve the integer RMP heuristics and to solve CG sub-problems); and *decomposition approaches* to initialize CG and also in the sub-problem resolution of CG. In such hybridization, mathematical programming is hybridized with the trajectory of local search metaheuristics. A perspective is also to investigate the collaboration between CG heuristics and population metaheuristics. The aggressive CG scheme and benefits or diversification highlighted in this paper offer perspectives to solve CG sub-problems and generate several columns with GA, PSO or ACO heuristics. Recent advances for these population meta-heuristics are promising for such applications [80–83].

Our instances are now public, and the graduations of difficulty and highly constrained instances with fewer possible neighbor moves are useful for challenging local search implementation. The first results with LocalSolver were already good, with some difficulties on highly constrained instances; we note that our instances are now in their benchmark instances to help the parametrization of their generic solver. This is a perspective for other generic codes with local search applied for vehicle routing problems.

Other perspectives are to extend these results with extensions of VRP and technician routing problems with more constraints (as in [18,19,34]) or considering multi-objective or robust optimization extensions, as in [84–86]. The matheuristic stabilization of CG algorithm may also be extended for other problems where CG algorithm is efficient, even without having heterogeneous sub-problems. It raises the question whether the matheuristic stabilization techniques proposed in this paper would also be efficient in the case of symmetrical sub-problems.

Author Contributions: Conceptualization, N.D. and E.-G.T.; Methodology, N.D. and E.-G.T.; Software, R.P; Validation, R.P.; Data curation, R.P.; Writing—original draft preparation, N.D.; Writing—review and editing, N.D and R.P.; Supervision, E.-G.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting reported results can be found at https://github.com/ndupin/SiteDepTRPTW (accessed on 25 October 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Williams, H.P. *Model Building in Mathematical Programming*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
2. Blum, C.; Puchinger, J.; Raidl, G.; Roli, A. Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput.* **2011**, *11*, 4135–4151. [CrossRef]
3. Jourdan, L.; Basseur, M.; Talbi, E.G. Hybridizing exact methods and metaheuristics: A taxonomy. *Eur. J. Oper. Res.* **2009**, *199*, 620–629. [CrossRef]
4. Dupin, N.; Talbi, E. Matheuristics to optimize refueling and maintenance planning of nuclear power plants. *J. Heuristics* **2021**, *27*, 63–105. [CrossRef]
5. Dupin, N.; Talbi, E. Parallel matheuristics for the discrete unit commitment problem with min-stop ramping constraints. *Int. Trans. Oper. Res.* **2020**, *27*, 219–244. [CrossRef]
6. Talbi, E. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Ann. Oper. Res.* **2016**, *240*, 171–215. [CrossRef]
7. Peschiera, F.; Dell, R.; Royset, J.; Haït, A.; Dupin, N.; Battaïa, O. A novel solution approach with ML-based pseudo-cuts for the Flight and Maintenance Planning problem. *Spectr.* **2021**, *43*, 635–664. [CrossRef]
8. Riedler, M.; Jatschka, T.; Maschler, J.; Raidl, G. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *Int. Trans. Oper. Res.* **2020**, *27*, 573–613. [CrossRef]
9. Dupin, N.; Talbi, E. Machine learning-guided dual heuristics and new lower bounds for the refueling and maintenance planning problem of nuclear power plants. *Algorithms* **2020**, *13*, 185. [CrossRef]
10. Toth, P.; Vigo, D. *Vehicle Routing: Problems, Methods, and Applications*; SIAM: Philadelphia, PA, USA, 2014; Volume 18,
11. Feillet, D. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR Q. J. Oper. Res.* **2010**, *8*, 407–424. [CrossRef]
12. Pessoa, A.; Sadykov, R.; Uchoa, E.; Vanderbeck, F. A generic exact solver for vehicle routing and related problems. *Math. Program.* **2020**, *183*, 483–523. [CrossRef]
13. Danna, E.; Le Pape, C. Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In *Column Generation*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 18, pp. 99–129.
14. Doerner, K.F.; Schmid, V. Survey: Matheuristics for rich vehicle routing problems. In *International Workshop on Hybrid Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 206–221.
15. Archetti, C.; Speranza, M.G. A survey on matheuristics for routing problems. *EURO J. Comput. Optim.* **2014**, *2*, 223–246. [CrossRef]
16. Furian, N.; O'Sullivan, M.; Walker, C.; Çela, E. A machine learning-based branch and price algorithm for a sampled vehicle routing problem. *OR Spectrum* **2021**, *43*, 693–732. [CrossRef]
17. Morabit, M.; Desaulniers, G.; Lodi, A. Machine-Learning-Based Column Selection for Column Generation. *Transp. Sci.* **2021**, *5*, 815–831. [CrossRef]
18. Dutot, P.F.; Laugier, A.; Bustos, A.M. *Technicians and Interventions Scheduling for Telecommunications*; Technical Report for France Telecom R&D: Paris, French, 2006.
19. Gromicho, J.; van't Hof, P.; Vigo, D. The VeRoLog verolog solver challenge 2019. *J. Veh. Rout. Algorithms* **2019**, *2*, 109–111. [CrossRef]
20. Cordeau, J.F.; Laporte, G.; Mercier, A. A unified tabu search heuristic for vehicle routing problems with time windows. *J. Oper. Res. Soc.* **2001**, *52*, 928–936. [CrossRef]
21. Cordeau, J.F.; Laporte, G. A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR Inf. Syst. Oper. Res.* **2001**, *39*, 292–298. [CrossRef]
22. Polacek, M.; Hartl, R.F.; Doerner, K.; Reimann, M. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *J. Heuristics* **2004**, *10*, 613–627. [CrossRef]
23. Feillet, D.; Dejax, P.; Gendreau, M.; Gueguen, C. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* **2004**, *44*, 216–229. [CrossRef]

24. Cappanera, P.; Gouveia, L.; Scutellà, M.G. Models and valid inequalities to asymmetric skill-based routing problems. *EURO J. Transp. Logist.* **2013**, *2*, 29–55. [CrossRef]
25. Cappanera, P.; Requejo, C.; Scutellà, M.G. Temporal constraints and device management for the Skill VRP: mathematical model and lower bounding techniques. *Comput. Oper. Res.* **2020**, *124*, 105054. [CrossRef]
26. Pessoa, A.; Sadykov, R.; Uchoa, E. Enhanced Branch-Cut-and-Price algorithm for heterogeneous fleet vehicle routing problems. *Eur. J. Oper. Res.* **2018**, *270*, 530–543. [CrossRef]
27. Schwarze, S.; Voß, S. Improved load balancing and resource utilization for the skill vehicle routing problem. *Optim. Lett.* **2013**, *7*, 1805–1823. [CrossRef]
28. Schwarze, S.; Voß, S. A bicriteria skill vehicle routing problem with time windows and an application to pushback operations at airports. In *Logistics Management*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 289–300.
29. Yan, X.; Xiao, B.; Xiao, Y.; Zhao, Z.; Ma, L.; Wang, N. Skill vehicle routing problem with time windows considering dynamic service times and time-skill-dependent costs. *IEEE Access* **2019**, *7*, 77208–77221. [CrossRef]
30. Castillo-Salazar, J.and Landa-Silva, D.; Qu, R. Workforce scheduling and routing problems: Literature survey and computational study. *Ann. Oper. Res.* **2016**, *239*, 39–67. [CrossRef]
31. Kovacs, A.; Parragh, S.; Doerner, K.; Hartl, R. Adaptive large neighborhood search for service technician routing and scheduling problems. *J. Sched.* **2012**, *15*, 579–600. [CrossRef]
32. Xie, F.; Potts, C.; Bektaş, T. Iterated local search for workforce scheduling and routing problems. *J. Heuristics* **2017**, *23*, 471–500. [CrossRef]
33. Pillac, V.; Gueret, C.; Medaglia, A. A parallel matheuristic for the technician routing and scheduling problem. *Optim. Lett.* **2013**, *7*, 1525–1535. [CrossRef]
34. Mendoza, J.; Montoya, A.; Guéret, C.; Villegas, J. A parallel matheuristic for the technician routing problem with conventional and electric vehicles. In Proceedings of the 12th Metaheuristics International Conference, Las Palmas de Gran Canaria, Spain, 19–24 February 2017.
35. Chen, X.; Thomas, B.W.; Hewitt, M. The technician routing problem with experience-based service times. *Omega* **2016**, *61*, 49–61. [CrossRef]
36. Pillac, V.; Guéret, C.; Medaglia, A. A fast reoptimization approach for the dynamic technician routing and scheduling problem. In *Recent Developments in Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 347–367.
37. Anoshkina, Y.; Meisel, F. Interday routing and scheduling of multi-skilled teams with consistency consideration and intraday rescheduling. *EURO J. Transp. Logist.* **2020**, *9*, 100012. [CrossRef]
38. Bley, A.; Karch, D.; D'Andreagiovanni, F. WDM fiber replacement scheduling. *Electron. Notes Discret. Math.* **2013**, *41*, 189–196. [CrossRef]
39. Pugliese, L.; Guerriero, F. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks* **2013**, *62*, 183–200. [CrossRef]
40. Sadykov, R.; Uchoa, E.; Pessoa, A. A bucket graph—Based labeling algorithm with application to vehicle routing. *Transp. Sci.* **2021**, *55*, 4–28. [CrossRef]
41. Baldacci, R.; Battarra, M.; Vigo, D. Routing a heterogeneous fleet of vehicles. In *The Vehicle Routing Problem: Latest Advances and New Challenges*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 3–27.
42. Baldacci, R.; Mingozzi, A. A unified exact method for solving different classes of vehicle routing problems. *Math. Program.* **2009**, *120*, 347. [CrossRef]
43. Bettinelli, A.; Ceselli, A.; Righini, G. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transp. Res. Part C Emerg. Technol.* **2011**, *19*, 723–740. [CrossRef]
44. Gendreau, M.; Potvin, J.Y.; Bräysy, O.; Hasle, G.; Løkketangen, A. Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In *The Vehicle Routing Problem: Latest Advances and New Challenges*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 143–169.
45. Pisinger, D.; Ropke, S. A general heuristic for vehicle routing problems. *Comput. Oper. Res.* **2007**, *34*, 2403–2435. [CrossRef]
46. Vidal, T.; Crainic, T.; Gendreau, M.; Prins, C. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Comput. Oper. Res.* **2013**, *40*, 475–489. [CrossRef]
47. Rizzoli, A.; Montemanni, R.; Lucibello, E.; Gambardella, L. Ant colony optimization for real-world vehicle routing problems. *Swarm Intell.* **2007**, *1*, 135–151. [CrossRef]
48. Hurkens, C. Incorporating the strength of MIP modeling in schedule construction. *RAIRO-Oper. Res.* **2009**, *43*, 409–420. [CrossRef]
49. Mancini, S. A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transp. Res. Part C Emerg. Technol.* **2016**, *70*, 100–112. [CrossRef]
50. Taillard, E. A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO Oper. Res.* **1999**, *33*, 1–14. [CrossRef]
51. Vidal, T.; Crainic, T.; Gendreau, M.; Prins, C. A unified solution framework for multi-attribute vehicle routing problems. *Eur. J. Oper. Res.* **2014**, *234*, 658–673. [CrossRef]
52. Pekel, E. Solving technician routing and scheduling problem using improved particle swarm optimization. *Soft Comput.* **2020**, *24*, 19007–19015. [CrossRef]
53. Pereira, D.; Alves, J.; de Oliveira Moreira, M. A multiperiod workforce scheduling and routing problem with dependent tasks. *Comput. Oper. Res.* **2020**, *118*, 104930. [CrossRef]

54. Cortés, C.E.; Gendreau, M.; Rousseau, L.; Souyris, S.; Weintraub, A. Branch-and-price and constraint programming for solving a real-life technician dispatching problem. *Eur. J. Oper. Res.* **2014**, *238*, 300–312. [CrossRef]
55. Zamorano, E.; Stolletz, R. Branch-and-price approaches for the multiperiod technician routing and scheduling problem. *Eur. J. Oper. Res.* **2017**, *257*, 55–68. [CrossRef]
56. Mathlouthi, I.; Gendreau, M.; Potvin, J. Branch-and-price for a multi-attribute technician routing and scheduling problem. *SN Oper. Res. Forum* **2021**, *2*, 1–35. [CrossRef]
57. Penna, P.; Subramanian, A.; Ochi, L.S.; Vidal, T.; Prins, C. A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. *Ann. Oper. Res.* **2019**, *273*, 5–74. [CrossRef]
58. Graf, B. Adaptive large variable neighborhood search for a multiperiod vehicle and technician routing problem. *Networks* **2020**, *76*, 256–272. [CrossRef]
59. Jagtenberg, C.; Maclaren, O.; Mason, A.; Raith, A.; Shen, K.; Sundvick, M. Columnwise neighborhood search: A novel set partitioning matheuristic and its application to the VeRoLog Solver Challenge 2019. *Networks* **2020**, *76*, 273–293. [CrossRef]
60. Kheiri, A.; Ahmed, L.; Boyacı, B.; Gromicho, J.; Mumford, C.; Özcan, E.; Dirikoç, A. Exact and hyper-heuristic solutions for the distribution-installation problem from the VeRoLog 2019 challenge. *Networks* **2020**, *76*, 294–319. [CrossRef]
61. Kastrati, V.; Ahmeti, A.; Musliu, N. Solving Vehicle Routing and Scheduling with Delivery and Installation of Machines using ILS. In Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT, Bruges, Belgium, 24–27 August 2021; Volume 1.
62. Fırat, M.; Hurkens, C. An improved MIP-based approach for a multi-skill workforce scheduling problem. *J. Sched.* **2012**, *15*, 363–380. [CrossRef]
63. Pokutta, S.; Stauffer, G. France Telecom workforce scheduling problem: A challenge. *RAIRO Oper. Res.* **2009**, *43*, 375–386. [CrossRef]
64. Estellon, B.; Gardi, F.; Nouioua, K. High-Performance Local Search for Task Scheduling with Human Resource Allocation. In *International Workshop on Engineering Stochastic Local Search Algorithms*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–15.
65. Cordeau, J.F.; Laporte, G.; Pasin, F.; Ropke, S. Scheduling technicians and tasks in a telecommunications company. *J. Sched.* **2010**, *13*, 393–409. [CrossRef]
66. Hashimoto, H.; Boussier, S.; Vasquez, M.; Wilbaut, C. A GRASP-based approach for technicians and interventions scheduling for telecommunications. *Ann. Oper. Res.* **2011**, *183*, 143–161. [CrossRef]
67. Ascheuer, N.; Fischetti, M.; Grötschel, M. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks* **2000**, *36*, 69–79. [CrossRef]
68. Öncan, T.; Altınel, İ.K.; Laporte, G. A comparative analysis of several asymmetric traveling salesman problem formulations. *Comput.* **2009**, *36*, 637–654. [CrossRef]
69. Fox, K.; Gavish, B.; Graves, S. An n-Constraint Formulation of the (Time-Dependent) Traveling Salesman Problem. *Oper. Res.* **1980**, *28*, 1018–1021. [CrossRef]
70. Lindahl, M.; Sørensen, M.; Stidsen, T. A fix-and-optimize matheuristic for university timetabling. *J. Heuristics* **2018**, *24*, 645–665. [CrossRef]
71. Fischetti, M.; Lodi, A. Local branching. *Math Program.* **2003**, *98*, 23–47. [CrossRef]
72. Taillard, É.; Voss, S. POPMUSIC-Partial optimization metaheuristic under special intensification conditions. In *Essays and Surveys in Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 613–629.
73. Mancini, S. A combined multistart random constructive heuristic and set partitioning based formulation for the vehicle routing problem with time dependent travel time. *Comput. Oper. Res.* **2017**, *88*, 290–296. [CrossRef]
74. Queiroga, E.; Sadykov, R.; Uchoa, E. A POPMUSIC matheuristic for the capacitated vehicle routing problem. *Comput. Oper. Res.* **2021**, *136*, 105475. [CrossRef]
75. du Merle, O.; Villeneuve, D.; Desrosiers, J.; Hansen, P. Stabilized column generation. *Discret. Math.* **1999**, *94*, 229–237. [CrossRef]
76. Lazic, J.; Hanafi, S.; Mladenovic, N.; Urozevic, D. Variable Neighbourhood Decomposition Search for 0-1 Mixed Integer Programs. *Comput. Oper. Res.* **2010**, *37*, 1055–1067. [CrossRef]
77. Desaulniers, G.; Lessard, F.; Hadjar, A. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transp. Sci.* **2008**, *42*, 387–404. [CrossRef]
78. Desrosiers, J.; Lübbecke, M. A primer in column generation. In *Column Generation*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–32.
79. Dupin, N. Column generation for the discrete UC problem with min-stop ramping constraints. *IFAC-PapersOnLine* **2019**, *52*, 529–534. [CrossRef]
80. Vidal, T.; Crainic, T.; Gendreau, M.; Lahrichi, N.; Rei, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Oper. Res.* **2012**, *60*, 611–624. [CrossRef]
81. Wang, C.; Guo, C.; Zuo, X. Solving multi-depot electric vehicle scheduling problem by column generation and genetic algorithm. *Appl. Soft Comput.* **2021**, *112*, 107774. [CrossRef]
82. Wang, F.; Zhang, H.; Li, K.; Lin, Z.; Yang, J.; Shen, X. A hybrid particle swarm optimization algorithm using adaptive learning strategy. *Inf. Sci.* **2018**, *436*, 162–177. [CrossRef]
83. Nurcahyadi, T.; Blum, C. Adding Negative Learning to Ant Colony Optimization: A Comprehensive Study. *Mathematics* **2021**, *9*, 361. [CrossRef]

84. Agra, A.; Christiansen, M.; Figueiredo, R.; Hvattum, L.; Poss, M.; Requejo, C. The robust vehicle routing problem with time windows. *Comput. Oper. Res.* **2013**, *40*, 856–866. [CrossRef]
85. Jozefowiez, N.; Semet, F.; Talbi, E. Multi-objective vehicle routing problems. *Eur. J. Oper. Res.* **2008**, *189*, 293–309. [CrossRef]
86. Glize, E.; Jozefowiez, N.; Ngueveu, S. An ε-constraint column generation-and-enumeration algorithm for Bi-Objective Vehicle Routing Problems. *Comput. Oper. Res.* **2021**, *138*, 105570. [CrossRef]