A parallel matheuristic for the technician routing and scheduling problem

Victor Pillac, Christelle Gueret, Andrés Medaglia

A Parallel Matheuristic for the Technician Routing and Scheduling Problem

Victor PILLAC[1,2], Christelle GUÉRET*[,3], and Andrés L. MEDAGLIA[2]

[1]*LUNAM Université, École des Mines de Nantes, IRCCyN UMR 6597, Nantes, France*
[2]*Universidad de Los Andes, COPA & CEIBA, Bogotá, Colombia*
[3]*LUNAM Université, LISA - IUT Angers-Cholet, Angers, France*

September 2012

The present work has been accepted for publication in *Optimization Letters*
doi:10.1007/s11590-012-0567-4
The orginal publication is available at springer.com

Abstract

The Technician Routing and Scheduling Problem (TRSP) consists in routing staff to serve requests for service, taking into account time windows, skills, tools, and spare parts. Typical applications include maintenance operations and staff routing in telecoms, public utilities, and in the health care industry. In this paper, we present a formal definition of the TRSP, discuss its relation with the Vehicle Routing Problem with Time Windows (VRPTW), and review related research. From a methodological perspective, we describe a matheuristic composed of a constructive heuristic, a parallel Adaptive Large Neighborhood Search (pALNS), and a mathematical programming based post-optimization procedure that successfully tackles the TRSP. We validate the matheuristic on the Solomon VRPTW instances, where we achieve an average gap of 0.23%, and matched 44 out of 55 optimal solutions. Finally, we illustrate how the matheuristic successfully solves a set of TRSP instances extended from the Solomon benchmark.

1 Introduction

The Technician Routing and Scheduling Problem (TRSP) deals with a limited crew of technicians $\mathscr{K}$ that serves a set of requests $\mathscr{R}$. In the TRSP, each technician has a set of skills, tools, and spare parts, while requests require a subset of each. The problem is then to design a set of tours of minimal total duration such that each request is fulfilled exactly once, within its time window, by a technician with the required skills, tools, and spare parts. It is important to note that the departure of technicians may be delayed

*Corresponding author: gueret@mines-nantes.fr

to minimize the waiting time at each visited request, thus reducing the duration of tours. The TRSP naturally arises in a wide range of settings, including telecoms, public utilities, and companies planning maintenance operations. The TRSP can be seen as an extension of the Vehicle Routing Problem with Time Windows (VRPTW), where technicians play the role of vehicles and requests are made by clients. Thus, it belongs to the class of NP-Hard problems.

A distinctive feature of this problem is the presence of compatibility constraints between technicians and requests. While skills are intrinsic attributes, technicians may carry different tools and spare parts over the planning horizon. Technicians start their tour from their home, with a set of tools and spare parts that allows them to serve an initial set of requests. They also have the opportunity to replenish their tools and spare parts at a central depot at any time to serve more requests. Tools can be seen as renewable resources, while spare parts are non-renewable and consumed once the technician serves a request.

The remainder of this paper is organized as follows: Section 2 reviews the literature on problems related to the TRSP; Section 3 introduces the proposed matheuristic; Section 4 presents experimental results; and finally, Section 5 concludes this work and outlines directions for future research.

2 Literature review

The technician scheduling problem is closely related to the TRSP, but does not consider the routing aspects, nor the tool and spare part constraints. It was featured in the 2007 French Operational Research Society (ROADEF) challenge. We refer the reader to the work by Cordeau et al. [4] and Hashimoto et al. [6] for two solution approaches to a multi-day variant in which teams are assembled to serve requests. Kovacs et al. [7] studied an extension of this problem, namely, the Service Technician Routing and Scheduling Problem (STRSP), which considers routing costs, skills, and team building.

Bredström and Rönnqvist [3] present a generic mixed integer programming formulation for a Vehicle Routing and Scheduling Problem with Time Windows (VRSPTW) in which some clients must be visited simultaneously by two or more vehicles. The authors do not explicitly consider skills, but the proposed model accounts for compatibility constraints between vehicles and requests. Parragh [8] also tackled a variant with synchronization between technician visits.

A practical consideration in technician routing is that it may not be possible or desirable to serve all requests. Xu and Chiu [20] studied a variant of the TRSP in which the objective is to maximize the number of requests served while accounting for skill constraints and request urgency. Tang et al. [16] also considered requests with different urgency levels. The authors use a multi-period maximum collection problem formulation with time-dependent rewards modeling customer preferences. Tsang and Voudouris [17] solved a problem faced by British Telecom where technician skills affect the time required to serve a request.

Finally, home care routing and scheduling problems are related to the TRSP in the sense that they consider patients that need to be visited by staff with specific skills and

within a given time frame. We refer the interested reader to the case studies by Bertels and Fahle [2], Eveborn et al. [5], and Akjiratikarl et al. [1].

In summary, technician routing problems have received limited attention and to the best of our knowledge, no work considers tools or spare parts, two important components of real-world applications. The present work, based on a real problem, addresses this aspect and proposes a parallel matheuristic approach for the TRSP.

3 The proposed matheuristic

This section outlines the proposed matheuristic that comprises a fast constructive heuristic, a parallel adaptive large neighborhood search, and a mathematical programming based post-optimization.

3.1 Regret constructive heuristic

Regret heuristics [11] are constructive heuristics that incorporate a look ahead component. At each iteration the algorithm inserts the request with the greatest *regret* value at the best position, where the regret value is an estimation of the additional cost incurred if a request is not inserted at its best position.

More formally, let $\mathscr{U}$ be the set of requests to be inserted and δ_i^k be the cost of inserting request i at its best position in its k-th best route. The *regret-q* heuristic inserts at its best position request $i^* = \arg\max_{i \in \mathscr{U}} \left\{ \sum_{k=2}^{q} \left(\delta_i^k - \delta_i^1 \right) \right\}$ (ties are broken by choosing the request with the lowest δ_i^1 value). It is worth noting that *regret-1* corresponds to the well-known best insertion heuristic.

When evaluating the insertion of a request in a tour we need to consider the possibility to plan a trip to the main depot to pick up additional tools and spare parts. The procedure first checks for the best feasible insertion without considering trips to the depot. If no feasible insertion is found, it then considers each possible combination of request and main depot insertions. Insertion feasibility and cost are evaluated in constant time using the concepts of *waiting time* and *forward time slack* introduced by Savelsbergh [13].

We use a *regret-3* heuristic to design an initial set of K solutions that will then be improved by the parallel adaptive large neighborhood search.

3.2 Parallel Adaptive Large Neighborhood Search

Shaw [14] introduced the Large Neighborhood Search algorithm (LNS), which works by successively *destroying* and *repairing* a current solution. Pisinger and Ropke [10] extended LNS by using several destroy and repair operators and adding an adaptive layer to select them, leading to the Adaptive LNS algorithm (ALNS). In this work, we propose a parallel version of ALNS, namely pALNS, that takes advantage of parallel architectures to achieve significant speedups.

Algorithm 1 presents the outline of pALNS. The algorithm maintains a pool $\mathscr{P}$ of N promising solutions that are optimized in K subprocesses (note that $N \geq K$). For each *master* iteration, a subset of K promising solutions is selected randomly (line 4) and

Algorithm 1 Parallel Adaptive Large Neighborhood Search (pALNS) algorithm

Input: $\mathscr{P}$, initial solutions; z, evaluation function; Θ^-/Θ^+, set of destroy/repair operators; N, maximum size of the solution pool; K, number of subprocesses; I^m, number of master iterations; I^p, number of iterations performed in parallel.

Output: Π^*, the best solution found; Ω', the pool of tours for the post-optimization.

$\Omega' \leftarrow \emptyset$
$\Pi^* \leftarrow \arg\min_{\Pi \in \mathscr{P}} \{z(\Pi)\}$
for I^m iterations **do**
  $\mathscr{P}' \leftarrow \texttt{selectSubset}(\mathscr{P}, K)$ ▷ Select a subset of K solutions
  parallel forall Π **in** $\mathscr{P}'$ **do**
    $\Pi^p \leftarrow \Pi$ ▷ Current solution for this subprocess
    for I^p iterations **do**
      $d \leftarrow \texttt{select}(\Theta^-); r \leftarrow \texttt{select}(\Theta^+)$ ▷ Select destroy/repair
      $\Pi' \leftarrow \texttt{r}(\texttt{d}(\Pi^p))$ ▷ Destroy and repair current solution
      if $\texttt{accept}(\Pi', \Pi^p)$ **then**
        $\Pi^p \leftarrow \Pi'$ ▷ Π' is accepted as current solution
      end if
      if $z(\Pi') < z(\Pi^*)$ **then**
        $\Pi^* \leftarrow \Pi'$ ▷ Π' is the best solution found so far
      end if
      $\texttt{updateScore}(d, r, \Pi')$ ▷ Update d and r scores
      $\Omega' \leftarrow \Omega' \cup \{\vec{\pi}\}_{\vec{\pi} \in \Pi'}$ ▷ Add tours from Π' to the set-covering tour pool Ω'
    end for
    $\mathscr{P} \leftarrow \mathscr{P} \cup \{\Pi^p\}$ ▷ Add Π^p to the pool $\mathscr{P}$
  end forall
  $\mathscr{P} \leftarrow \texttt{retain}(\mathscr{P}, \Pi^*, N)$ ▷ Retain at most N solutions in the pool $\mathscr{P}$
end for
return Π^*, Ω'

distributed among independent subprocesses. Then for I^p iterations, each subprocess selects destroy and repair operators with a roulette wheel mechanism that adaptively reflects their past performance (line 8). The current solution is then successively destroyed and repaired, producing a temporary solution (line 9). The temporary solution is either accepted as the subprocess current solution or rejected according to a simulated annealing criterion (line 10) The weights of the destroy and repair operators are updated depending on their performance (line 16) and the tours from the solution are stored for the post-optimization (line 17). The final current solution of each subprocess is added to the pool of promising solutions (line 19). When all subprocesses have terminated, a filtering procedure ensures that the pool contains at most N solutions, including the best solution found so far (line 21). The algorithm stops after I^m master iterations, which corresponds to $I = I^m \times I^p \times K$ ALNS iterations. What follows is a detailed description of the main components of pALNS.

3.2.1 Destroy

Destroy operators remove a random number of requests from the current solution. We used three destroy operators originally proposed by Pisinger and Ropke [10]: *random*, *critical*, and *related*. The random destroy operator removes requests randomly from their current tours; the critical destroy operator removes requests that are among the most costly in the current solution; finally, the related destroy removes requests that share common characteristics by first selecting a seed request, and then removing related requests. It is important to note that all three destroy operators are randomized.

We propose two relatedness metrics tailored for the TRSP that define two new destroy operators. The *a priori relatedness* is a precalculated metric that does not depend on the current position of the requests in the tours and combines three components: geographic distance, difference of due dates, and number of technicians that can serve both requests. On the other hand, *time relatedness* measures the difference between the service time of two requests in the current solution.

3.2.2 Repair

Repair operators attempt to insert requests that are currently unserved. If requests cannot be reinserted, a penalty proportional to the number of unserved requests is added to the objective function. This penalty approach allows infeasible solutions to be considered as the current solution during the search, and can be interpreted as the possible outsourcing of some requests. Our implementation is based on three repair heuristics: *best insertion*, *regret-2*, and *regret-3*.

3.2.3 Adaptive layer

At each iteration, the pALNS algorithm selects a destroy and a repair operator using a roulette wheel mechanism. Operator θ is selected with probability w_θ. Let $\Theta^\cdot$ be either the set of destroy (Θ^-) or repair (Θ^+) operators. As in the original ALNS algorithm, probabilities are initialized with value $\frac{1}{|\Theta^\cdot|}$. However, they are then updated every l iterations as follows: $w_\theta \leftarrow (1-\rho)w_\theta + \rho\frac{s_\theta}{\sum_{\theta\in\Theta^\cdot} s_\theta}$, where $\rho \in [0,1]$ is the *reaction*

factor which defines how quickly probabilities are adjusted, and s_θ is the *score* of operator θ in the last l iterations. Note that this formula ensures that $\sum_{\theta\in\Theta} w_\theta = 1$ at all time. The scores s_θ are maintained at the master level. They are reset to 0 every l iterations and updated at the end of each iteration depending on the new solution Π': a score of σ_1 is granted for a new best solution, σ_2 for an improving solution, σ_3 for a non-improving but accepted solution, and σ_4 for a rejected solution.

3.2.4 Acceptance criterion

The pALNS algorithm relies on a simulated annealing acceptance criterion: a new solution Π' is accepted with probability $e^{\frac{z(\Pi)-z(\Pi')}{T}}$, where T is the *temperature* parameter. T is initialized with value T_0 and reduced at each iteration by a *cooling factor* c. Parameters T_0 and c are fixed depending on the initial solution and the target number of iterations [10].

3.2.5 Promising solution pool

The solution pool acts as a shared memory and allows subprocesses to collaborate efficiently. The method `retain` ensures that $\mathscr{P}$ contains at most N solutions: if $|\mathscr{P}| > N$ then the method retains the N best solutions according to the fitness function $f(\Pi) = \text{rank}_z(\Pi) + \text{rank}_d(\Pi)$, where $\text{rank}_z(\Pi)$ is the rank of solution Π according to its objective value and $\text{rank}_d(\Pi)$ is the rank of Π according to a diversity metric. For the latter metric, we use the average broken pairs distance [12] to measure the diversity of solution Π relative to the other solutions in $\mathscr{P}$. This fitness function is inspired by the *biased fitness* introduced by Vidal et al. [18] in a genetic algorithm with diversity management. It allows the preservation of solutions that are both diverse and promising in terms of cost. In addition, we ensure that $\mathscr{P}$ always contains the best solution found so far.

3.3 Set-covering based post-optimization

The pALNS algorithm generates one solution per ALNS iteration, but only keeps the best one. However, good solutions may contain poor tours, and conversely poor solutions may contain good tours. The proposed approach overcomes this limitation by solving a Set Covering model (SC) that combines the tours generated throughout the search to assemble a better solution. Note that a similar approach was for instance used by Villegas[19] to solve the Truck and Trailer Routing Problem (TTRP) showing excellent results.

3.3.1 Tour pool

Throughout the pALNS algorithm, we store in a pool Ω' the tours $\vec{\pi}$ that make up the temporary solutions Π' found by the algorithm (see Algorithm 1, line 17). Tours are either stored in a single hash table when solving the CVRPTW, or in a separate hash table per technician for the TRSP. We associate a 32-bit integer to each tour using

the hash function $\text{hash}(\vec{\pi}) = \oplus_{i \in \vec{\pi}} R[i]$, where R is an array associating a random 32-bit integer to each request and $\oplus$ is the `XOR` bit-wise operator. It is important to note that this hash function only considers the subset of requests in tour $\vec{\pi}$, ignoring their sequence which is not relevant for the set-covering model. Preliminary experiments revealed that the probability of having a hash collision was under 10^{-3}. Therefore, we ignore hash collisions and always keep the tour with the lowest cost, without checking if tours actually contain the same requests.

3.3.2 Mathematical model

Let $\Omega'_k \subseteq \Omega'$ be the subset of tours associated with technician k, c_t be the duration of tour t, and a_{ti} a binary parameter that takes the value of 1 if tour t visits request i and 0 otherwise. We denote by x_t a decision variable that takes the value of 1 if tour t is selected, and 0 otherwise. We can then formulate the TRSP on the subset Ω' of all feasible tours as follows:

$$\min \quad \sum_{t \in \Omega'} c_t x_t \tag{1}$$

$$s.t., \quad \sum_{t \in \Omega'} a_{ti} \cdot x_t \geq 1 \qquad \forall i \in \mathscr{R} \tag{2}$$

$$\sum_{t \in \Omega'_k} x_t \leq 1 \qquad \forall k \in \mathscr{K} \tag{3}$$

$$x_t \in \{0,1\} \qquad \forall t \in \Omega' \tag{4}$$

where the objective (1) minimizes the total routing duration, constraints (2) ensure that each request is served at least once, and constraints (3) guarantee each technician performs at most one tour.

Considering that requests must be served exactly once, one could argue that a set-partitioning formulation fits better. However, our model only contains a reduced subset of tours (columns), and therefore, we might not be able to find a good combination of tours that visit all requests exactly once. The drawback of this formulation is that the solution may visit a request more than once. In such event, the solution is repaired by removing the most costly duplicated visits.

4 Computational results

In this section we report computational results for the proposed matheuristic. All experiments were run using Java 7 and Gurobi 4.60 on an Ubuntu 11.10 64-bit machine, with an Intel i7 860 processor (4×2.8GHz) and 6GB of RAM, using $K = 8$ subprocesses. The pALNS algorithm was run for 25600 iterations ($I^p = 100$, $I^m = 32$) and a time limit of 30 minutes was enforced for the set-covering model. Because the destroy operators are randomized, pALNS is a non-deterministic algorithm, therefore we run it 10 times for each instance. The detailed parameter settings are shown in [9].

4.1 Validation on the VRPTW

The TRSP being a natural extension of the VRPTW, we validate our matheuristic on the 56 VRPTW instances from the Solomon benchmark [15]. The instances contain 100 requests located randomly (R), in clusters (C), or combining both (RC); with either a short (type 1) or long (type 2) planning horizon. These instances are organized combining location and horizon (i.e., C1, C2, R1, R2, RC1, and RC2), each group containing between 8 and 12 instances. For the VRPTW, we consider the minimization of the traveled distance[1] and replace constraints (3) from the set covering model by $\sum_{t\in\Omega} x_t \leq 25$ to model the 25-vehicle homogeneous fleet defined in the Solomon instances [15].

	Improvement		**Gap to BKS/Opt**		**Best known solutions**		**Time (s)**		
Group	Δ_{pALNS}	Δ_{SC}	pALNS	pALNS+SC	#Opt.	#BKS	pALNS	SC	$\lvert\Omega'\rvert$
C1	37.89%	0.00%	0.00%	0.00%	9/9	-	14.6	0.4	11550
C2	26.41%	0.02%	0.02%	0.00%	8/8	-	26.5	0.2	3479
R1	24.28%	0.44%	0.59%	0.14%	10/12	-	13.1	27.2	27303
R2	32.21%	0.25%	0.76%	0.51%	5/10	1/1	24.5	2.1	14161
RC1	25.06%	1.21%	1.38%	0.15%	6/8	-	12.6	25.1	25327
RC2	36.56%	0.43%	0.99%	0.55%	6/8	-	21.3	1.3	11822
All	**30.20%**	**0.38%**	**0.62%**	**0.23%**	**44/55**	**1/1**	**18.6**	**10.1**	**16293**

Table 1: Computational results for the Solomon [15] instances (average over 10 runs).

Table 1 summarizes the average results for each instance group. The first column defines the instance group, the second column contains the relative improvement between the initial solution and the solution returned by pALNS (Δ_{pALNS}), the third column reports the relative improvement between the pALNS solution and the pALNS+SC solution (Δ_{SC}). The fourth and fifth columns contain the average gap to the optimal or best known solution for pALNS and pALNS+SC. The sixth column reports the number of optimal solutions found (Opt.) over the number of known optimal solutions, while the seventh column reports the number of best known solutions (BKS) found over the number of heuristic BKS. Columns eight and nine show the average computational times for the pALNS and SC, and the last column reports the average size of the tour pool.

The overall average gap for pALNS+SC is just 0.23%, while Pisinger and Ropke [10] report a value of 0.36% using an ALNS with a larger number of destroy and repair operators[2]. This illustrates the importance of the post-optimization step of the matheuristic, which is able to divide the gap by a factor of 3.4 in 10s on average. On the other hand, the parallelization of the algorithm allowed for speedups of 3.5 times relative to a sequential implementation, leading to running times of 19s on average.

[1] Note that we truncate the distances to one decimal, as it is common practice when solving the Solomon instances [15] with the distance minimization as solely objective.

[2] In addition, it is important to note that 7 optimal solutions were not known at the time of their study, using the same values the average gap for our approach is of 0.16%.

4.2 Results on the TRSP

After validating our algorithmic building blocks on the VRPTW, in this section we analyze the performance of our matheuristic on randomly generated instances of the TRSP. Our testbed is composed of 56 instances of the TRSP based on the Solomon [15] benchmark. For each instance, we considered a crew of 25 technicians with different home locations, skills, initial set of tools and spare parts. In addition, we generated requests by adding skill, tool, and spare part information to each customer. These instances and our detailed solutions are publicly available at [9].

	Improvement	**Gap to BKS**		**Time (s)**		
Group	Δ_{SC}	pALNS	pALNS+SC	pALNS	SC	$\|\Omega\|$
C1	0.97%	1.22%	0.23%	24.0	388.9	67020
C2	0.35%	0.78%	0.42%	27.8	23.6	39334
R1	3.62%	4.96%	0.82%	28.9	500.2	30783
R2	0.23%	1.69%	1.46%	31.0	42.1	24396
RC1	3.06%	3.90%	0.68%	27.9	185.8	18638
RC2	0.49%	1.93%	1.43%	27.9	15.6	16917
All	**1.53%**	**2.54%**	**0.86%**	**28.1**	**210.1**	**32858**

Table 2: Computational results for 56 randomly generated TRSP instances.

Table 2 reports our results for the six groups of instances. Note that in this case we do not report the improvement of pALNS over the initial solution as the regret heuristic is not always able to insert all requests. In addition, the third and fourth columns report average gap to the best solution found in our experiments.

The SC post-optimization improves by 1.5% the pALNS solution, which is larger than the 0.38% improvement found for the VRPTW. This can be explained by the fact that the TRSP is harder for pALNS than the VRPTW, so further improvements can be found in the post-optimization phase. It is worth noting that on average the tour pool contains twice as many tours as in the VRPTW experiments. This can be explained by the fact that in the TRSP identical tours may be associated with different technicians. However the problem being overly constrained, it expectedly admits fewer feasible tours. In terms of running times, the post-optimization engine requires 20 times more computational effort to solve the TRSP than the VRPTW. This is due to the larger size of the tour pool and the presence of resource constraints (3) that destroy the set-covering structure, thus demanding more effort from the linear optimization engine which is likely to embed specific heuristics for pure set-covering models.

5 Conclusions and research perspectives

In this study we introduced a new challenging routing problem with numerous applications, namely the Technician Routing and Scheduling Problem. Distinctive features of this problem are the presence of compatibility constraints between technicians and requests; an initial set of tools and spare parts available to the technicians; the possibility for technicians to visit a main depot to pick up additional tools and spare parts; and the scheduling aspects introduced by the objective of minimizing the total tour duration.

We proposed a parallel matheuristic, which comprises three components: a regret constructive heuristic, a parallel adaptive large neighborhood search (pALNS), and a set-covering post-optimizer (SC). The parallelization of the ALNS allows a speed increase by a factor of 3.4 on a quad-core computer, while the post-optimization phase assembles a better solution by using tours gathered during the search. The resulting matheuristic maintains the flexibility of the ALNS, while improving its performance and reducing the need for complex operators.

We validated and measured the performance of the proposed matheuristic on the Solomon VRPTW benchmark, showing a negligible gap of 0.23% to the optimal and best known solutions (BKS), and finding 44 of the 55 optimal solutions in under 30s. Results on randomly generated instances of the TRSP illustrate the improvement that pALNS and SC bring over a constructive heuristic solution.

Future work will focus on the extension of the problem to a dynamic setting, in which unexpected delays and new requests may occur. To this end, we are focusing our research efforts on developing fast optimization procedures able to react in real time to changes in the problem information.

Acknowledgements Financial support for this work was provided by the CPER (Contrat de Projet Etat Region) Vallée du Libre (France); and the Centro de Estudios Interdisciplinarios Básicos y Aplicados en Complejidad (CEIBA, Colombia). This support is gratefully acknowledged. The authors would also like to thank Olivier Péton from the Ecole des Mines de Nantes and the anonymous reviewers for their insightful comments and suggestions.

References

[1] Akjiratikarl, C., Yenradee, P., Drake, P.R.: PSO-based algorithm for home care worker scheduling in the UK. Computers & Industrial Engineering **53**(4), 559 – 583 (2007)

[2] Bertels, S., Fahle, T.: A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. Computers & Operations Research **33**(10), 2866 – 2890 (2006)

[3] Bredström, D., Rönnqvist, M.: Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research **191**(1), 19–31 (2008)

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

[5] Eveborn, P., Flisberg, P., Ronnqvist, M.: LAPS CARE - an operational system for staff planning of home care. European Journal of Operational Research **171**(3), 962–976 (2006)

[6] Hashimoto, H., Boussier, S., Vasquez, M., Wilbaut, C.: A GRASP-based approach for technicians and interventions scheduling for telecommunications. Annals of Operations Research **183**, 143–161 (2011)

[7] Kovacs, A., Parragh, S., Doerner, K., Hartl, R.: Adaptive large neighborhood search for service technician routing and scheduling problems. Journal of Scheduling pp. 1–22 (2011). DOI 10.1007/s10951-011-0246-9

[8] Parragh, S.N.: Solving a real-world service technician routing and scheduling problem. In: Proceedings of the Seventh Triennial Symposium on Transportation Analysis (TRISTAN VII) (2010)

[9] Pillac, V., Guéret, C., Medglia, A.L.: A parallel matheuristic for the technician routing and scheduling problem: supplementary material (online) (2011). URL `http://hdl.handle.net/1992/1145`

[10] Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers & Operations Research **34**(8), 2403–2435 (2007)

[11] Potvin, J.Y., Rousseau, J.M.: A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research **66**(3), 331 – 340 (1993)

[12] Prins, C.: Two memetic algorithms for heterogeneous fleet vehicle routing problems. Engineering Applications of Artificial Intelligence **22**(6), 916–928 (2009)

[13] Savelsbergh, M.: The vehicle routing problem with time windows: minimizing route duration. INFORMS Journal on Computing **4**(2), 146–154 (1992)

[14] Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming – CP98, *Lecture Notes in Computer Science*, vol. 1520, pp. 417–431 (1998)

[15] Solomon, M.M.: Algorithms for the vehicle-routing and scheduling problems with time window constraints. Operations Research **35**(2), 254–265 (1987)

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

[17] Tsang, E., Voudouris, C.: Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Operations Research Letters **20**(3), 119–127 (1997)

[18] 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. Tech. Rep. 2011-61, CIRRELT (2011)

[19] Villegas, J.G.: Vehicle routing problems with trailers. 4OR: A Quarterly Journal of Operations Research (2012). DOI 10.1007/s10288-011-0186-4

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