Solving a Wind Turbine Maintenance Scheduling Problem

Aurélien Froger
Michel Gendreau
Jorge E. Mendoza
Eric Pinson
Louis-Martin Rousseau

February 2016

CIRRELT-2016-10

Solving a Wind Turbine Maintenance Scheduling Problem

Aurélien Froger[1,2,*], Michel Gendreau[2], Jorge E. Mendoza[3], Éric Pinson[2], Louis-Martin Rousseau[2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

[2] L'UNAM, Université Catholique de l'Ouest, LARIS EA 7315, Angers,France

[3] Université François-Rabelais de Tours, CNRS, LI EA 6300, OC ERL CNRS 6305, Tours, France

Abstract. Driven by climate change mitigation efforts, wind energy has significantly increased in recent years. In this context, it is essential to make its exploitation cost-effective. Maintenance of wind turbines therefore playsan essential role reducing breakdowns and ensuring high productivity levels. In this paper we discuss a challenging maintenance scheduling problem rising in wind farms. While the research in the field primarily focuses on condition-based maintenance strategies, we aim to address the problem on a short-term horizon considering fine-grained resource management. The objective is to find a maintenance plan that maximizes the production of the turbines while taking into account wind predictions, multiple task execution modes, and task-technician assignment constraints. To solve this problem, we propose a constraint programming-based large neighborhood search approach. We also propose two integer linear programming formulations that we solve using a commercial solver. We report results on randomly generated instances built with input from wind forecasting and maintenance scheduling software companies.

Keywords. Maintenance, scheduling, large neighborhood search, constraint programming.

Acknowledgements. This work was supported by Angers Loire Métropole through its research grant program; and by the Natural Sciences and Engineering Research Council of Canada (NSERC) through a grant that enabled the collaboration with the Canadian company WPred, which we would like to thank for their expertise. This support is gratefully acknowledged.

* Corresponding author: Aurélien.Froger@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2016

Introduction

With a 52-Gigawatts (GW) increase in the global installed capacity in 2014 (and a total of about 370 GW), wind energy is currently the world's fastest-growing source of electricity[1]. Boosted by the ever-increasing environment awareness and the constantly-decreasing cost of turbines, wind power is expected to account for up to 20% of the global electricity production by 2050[1] (vs. 2.4% in 2015). The Paris Agreement (resulting from the 2015 United Nations Climate Change Conference - COP21) is in this respect a clear evidence that the renewable energy sector will keep growing in order to reduce greenhouse gas emissions. In this context, efficient wind turbine maintenance planning and scheduling becomes a critical tool to prevent unnecessary downtime and excessive operational costs.

Maintenance planning and scheduling has been widely studied in different industrial contexts (see for example Budai et al. (2008) for a survey). In general, however, solutions remain sector-specific. In the particular case of traditional electricity generation plants, the problem is concerned with the definition of time intervals for preventive maintenance of generating units under financial (cost minimization, profits maximization) and/or reliability (leveling, maximization of the net reserves) considerations. By now, the literature reports on a number of solution approaches to tackle these problems. We refer the reader to Froger et al. (2016) for a comprehensive review. Unfortunately, these approaches are inapplicable to the wind power industry. One of the main reasons is that wind farms are usually owned by investment funds, and the operation and the maintenance of the turbines are often outsourced to a third party. As it stands, the stakeholders and the contractors face potentially conflicting objectives: maximize energy production vs. minimize maintenance costs. Another specificity of the wind power industry is that maintenance decisions are not correlated with the electricity demand since producers are not required to satisfy production goals. The objective is then to maximize the efficiency of the turbines. Last but not least, the wind power production is inherently volatile, and the meteorological conditions have a great impact on the maintenance plan.

Maintenance optimization for wind turbines has only recently started to received attention in the literature (we refer the reader to Ding et al. (2013) for a survey). This stream of research primarily focuses on the definition of maintenance policies according to failure models or/and condition monitoring. For instance, Carlos et al. (2013) proposed a stochastic model to find a maintenance strategy minimizing the operational costs and maximizing the annual energy production while taking into account meteorological conditions, degradation, and failures. The underlying methodology uses the Nelder-Mead method. The results of the optimization process is a maintenance time interval for each turbine. However, the study considers only each turbine in isolation. Although existing studies precisely define time intervals during which the maintenance has to be performed in order to reduce the loss of energy production, they do not consider a fine-grained resource management. Therefore, the obtained results are used more as guidelines to define maintenance time windows, than as an actual maintenance schedule. It is worth noticing that these time windows may also be specified by service contracts (e.g., preventive maintenance has to be performed every six months on each turbine).

Fine-grained resource management implies, among others, considering a skilled workforce, coping with individual or global resource unavailability periods (e.g., vacations), and taking into account resource locations constraints. Dealing with these issues requires considering a short-term planning horizon. In this context existing studies do not provide a solution to the problem but do provide valuable input. Indeed, they allow planners to define the tasks to be performed during the planning horizon and to set the maintenance time window constraints. Considering fine-grained resource management not only leads to maintenance scheduling problems that are closer to reality but also allows to obtain detailed maintenance plans that can be used on a daily or weekly basis. Detailed plans provide more accurate estimates of turbine downtimes and loss of production; two metrics that can otherwise be underestimated leading to significant prediction errors. For instance, producing a maintenance plan in which no operation generates a loss of production (e.g., is scheduled during periods of wind speed under 4 m/s, which is too low to produce energy) can almost never be achieved in practice, since human resources are a major bottleneck.

[1]The Global Wind Energy Council - Global wind report annual market update 2014 - http://www.gwec.net/wp-content/uploads/2015/03/GWEC_Global_Wind_2014_Report_LR.pdf, last accessed: 2016-02-15

To our knowledge, (Kovács et al., 2011) is the only study considering fine-grained resource management while scheduling maintenance operations on wind turbines. These authors aimed to minimize lost production due to maintenance and failures. They introduced incompatibilities between pairs of tasks and managed the assignment of teams of skilled workers to tasks. They modeled the problem as a *integer linear program* (IP) and solved it using a commercial solver. They performed experiments on instances with up to 50 tasks.

In this paper, we introduce a maintenance scheduling problem with resource management rising in the wind power industry. Our problem differs from that introduced by Kovács et al. (2011) in several ways. First, we manage resources (i.e., technicians) individually rather than by teams. Second, we consider multiple task execution modes that impact the duration of tasks as well as the resource requirements (De Reyck et al., 1998). Third, we present an alternative way to consider technician transfer times by introducing location-based incompatibility constraints between tasks. The objective of this new problem is to maximize the profits generated by the total power production of the wind turbines. The work targets a short-term horizon as wind predictions can only be reliably established few days in advance. When the length of the time horizon increases, or when good quality forecasts are unavailable, we maximize the availability of the turbines as an alternative objective.

The contributions of this paper are twofold. First, we introduce a new maintenance scheduling problem that is especially relevant to the wind power industry. We formally define our problem using two different paradigms, namely, integer linear programming and constraint programming. Second, we introduce a constraint programming-based large neighborhood search to efficiently tackle the problem. The proposed approach uses destruction operators either specifically conceived for the problem or adapted from the literature. The repair operator consists in solving a constraint programming (CP) model with some fixed variables using branching strategies specially tailored for the problem. We also present a new acceptance criterion based on elitism and Metropolis. We report computational results on instances with up to 90 tasks, 3 different skills and 40-period time horizons.

The remainder of this paper is organized as follows. In Section 1 we describe the problem. In Sections 2 and 3, we introduce two integer linear programming formulations and a constraint programming formulation for the problem. In Section 4 we present our constraint programming-based large neighborhood search approach. In Section 5 we discuss computational experiments. Finally in Section 6 we present our conclusions and outline research perspectives.

1. Problem statement

In our maintenance scheduling problem, we consider a finite time horizon $\mathcal{T}$ partitioned in periods of identical length that span over several days from a set $\mathcal{D}$. We denote $\mathcal{T}_d$ the set of periods covered by day $d \in \mathcal{D}$. We denote as $\mathcal{W}$ a set of wind turbines geographically distributed over a set $\mathcal{L}$ of wind farms. We denote by $\mathcal{I}$ the set of tasks to be scheduled during the planning horizon. Each task $i \in \mathcal{I}$ is associated with a specific location, denoted as l_i. We define $\mathcal{I}_l = \{i \in \mathcal{I} | l_i = l\}$ as the set of tasks to perform at location $l \in \mathcal{L}$. To execute these tasks, we have a finite set $\mathcal{R}$ of human resources (hereafter referred to as technicians). Technicians master specific skills from a set $\mathcal{S}$. Technician skills are expressed by a boolean vector λ_r over $\mathcal{S}$ such that $\lambda_{rs} = 1$ if technician r masters skill $s \in \mathcal{S}$, and $\lambda_{rs} = 0$ otherwise.

During a day, a technician is allowed to perform tasks at locations that are sufficiently close to be reached without too much travel time. We thus introduce a binary parameter $\sigma_{ll'}$ taking value 1 if a technician is allowed to work at both locations $l, l' \in \mathcal{L}$ during the same day, and 0 otherwise. Note that $\sigma_{ll'} = \sigma_{l'l}$ for every couple of locations (l, l'). Each technician $r \in \mathcal{R}$ has an individual availability schedule expressed by a boolean vector π_r, with $\pi_r^t = 1$ if r is available at period $t \in \mathcal{T}$, and $\pi_r^t = 0$ otherwise. In the latter case, the location of technician r is fixed to $l_r^t \in \mathcal{L}$. This allows to take into account assignments to tasks that are not part of the optimization for external reasons (e.g., tasks that cannot be moved or/and that are performed along with external companies).

Technicians enjoy a *rest period* between two consecutive days in $\mathcal{D}$. A rest period can be a night, a weekend, or a holiday. Tasks can be preempted during the rest periods (i.e, a technician can start a task at the end of one day and complete it at the beginning of the next day). Although we exclude rest periods from set $\mathcal{T}$, we count in the objective function the loss of production generated by tasks overlapping these specific times.

Tasks may have different impact on the availability of the turbines. Some tasks shut down one (or more) turbine(s) since the task starts until the task ends. For instance, during the maintenance of a wind farm's substation[2] no turbine in the farm can produce energy. Some tasks shut down the turbines when the technicians are effectively working on the task but not necessarily during the rest periods if the task is preempted. This is the case for the majority of the preventive maintenance jobs, as well as for wind turbines retrofit. Other tasks do not have any impact on energy production (e.g., wind farm inspections). We model task impact on energy production using two parameters. Parameter b_{wi} takes the value of 1 if task $i \in \mathcal{I}$ shuts down turbine $w \in \mathcal{W}$ when technicians are effectively working on the task, and 0 otherwise. Parameter $\widetilde{b}_{wi}$ takes the value of 1 if task i additionally shuts down turbine w during the rest periods it overlaps, and 0 otherwise. In virtually all the cases, it must be noted that parameters b_{wi} and $\widetilde{b}_{wi}$ are equal to 0 if turbine w is not located at wind farm l_i.

Tasks can only be executed during some specific periods. These take into account maintenance periodicity time windows, spare parts availability, safety work conditions (e.g., a technician cannot perform certain tasks on a turbine when the wind is too strong), and external restrictions imposed by the operator and/or the owner of wind farms.

Furthermore, each task $i \in \mathcal{I}$ has a set $\mathcal{M}_i$ of execution modes. For each mode $m \in \mathcal{M}_i$, there is an associated task duration d_{im} and a number q_{im} of required technicians. Switching modes after starting the execution of a task is forbidden. Additionally, only technicians mastering a specific skill $s_i \in \mathcal{S}$ can work on task i. For the sake of clarity, we denote as $\mathcal{R}_i$ the set of technicians that can perform task i. Note that $\mathcal{R}_i = \{r \in \mathcal{R} | \lambda_{rs_i} = 1\}$. Moreover, a technician assigned to a task has to work on it from the beginning to the end, even if the task is preempted during one or multiple rest periods.

Some subsets of tasks cannot overlap (i.e., be executed in parallel). We define $ov(\mathcal{I})$ the set containing all subsets of non-overlapping tasks.

The objective of the problem is to determine a schedule that maximizes the profits generated by the wind farms while meeting the constraints described above. We denote g_w^t the profit generated by wind turbine $w \in \mathcal{W}$ if it can produce energy during period $t \in \mathcal{T}$. Similarly, we denote $\widetilde{g}_w^d$ the profit generated by wind turbine w if it can produce energy during the rest period following day $d \in \mathcal{D}$. These profit values are defined depending whether the objective is to maximize the production of the wind turbines or their availability. While the first objective requires reliable wind speed forecasts, the latter considers profit values to be constant. Outsourcing a task $i \in \mathcal{I}$ is allowed but penalized with a cost of o_i. To model the outsourcing of task i, we create an additional execution mode m_i^0 and we add it to $\mathcal{M}_i$. In this study we do not consider other maintenance costs: we assume that, as it is common in practice, technicians earn a fixed salary, and we disregard travel costs as they are insignificant. Appendix Appendix A summarizes the notation used in this paper.

2. Integer linear programming formulations

In the following subsections, we present two integer linear programming models for the problem. The first formulation is an immediate *natural* formulation, whereas the second one aggregates some decision variables leading to a more compact formulation.

2.1. Natural formulation

For the first formulation, we use the following decision variables:

$$x_{im} = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ is executed in mode } m \in \mathcal{M}_i, \\ 0 & \text{otherwise.} \end{cases}$$

[2]A wind farm substation collects the energy produced by all the turbines of the farm and distributes it through the grid

$$s_i^t = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ starts at } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{ri} = \begin{cases} 1 & \text{if technician } r \in \mathcal{R} \text{ is assigned to task } i \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases}$$

$$c_i^t = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ ends at } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$e_i^t = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ is executed at } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$u_i^d = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ is executed during day } d \in \mathcal{D}, \\ 0 & \text{otherwise.} \end{cases}$$

$$f_w^t = \begin{cases} 1 & \text{if turbine } w \in \mathcal{W} \text{ can produce energy at } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\widetilde{f}_w^d = \begin{cases} 1 & \text{if turbine } w \in \mathcal{W} \text{ can produce energy during the rest period following day } d \in \mathcal{D}, \\ 0 & \text{otherwise.} \end{cases}$$

$$z_{ri}^t = \begin{cases} 1 & \text{if technician } r \in \mathcal{R} \text{ is assigned to task } i \in \mathcal{I} \text{ during period } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$v_{rl}^t = \begin{cases} 1 & \text{if technician } r \in \mathcal{R} \text{ is at location } l \in \mathcal{L} \text{ during period } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

An intuitive formulation is defined as the following integer linear program $[P1]$:

$$[P1] \quad \max \sum_{w \in \mathcal{W}} \left(\sum_{t \in \mathcal{T}} g_w^t f_w^t + \sum_{d \in \mathcal{D}} \widetilde{g}_w^d \widetilde{f}_w^d \right) - \sum_{i \in \mathcal{I}} o_i x_{im_i^0} \tag{1}$$

subject to:

$$\sum_{m \in \mathcal{M}_i} x_{im} = 1 \quad \forall i \in \mathcal{I}, \tag{2}$$

$$e_i^0 = 0 \quad \forall i \in \mathcal{I}, \tag{3}$$

$$e_i^t = e_i^{t-1} + s_i^t - c_i^t \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, t > 0, \tag{4}$$

$$\sum_{t \in \mathcal{T}} s_i^t = 1 \quad \forall i \in \mathcal{I}, \tag{5}$$

$$\sum_{t \in \mathcal{T}} c_i^t = 1 \quad \forall i \in \mathcal{I}, \tag{6}$$

$$e_i^t \le \gamma_i^t \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, \tag{7}$$

$$\sum_{i \in \mathcal{B}} e_i^t \le 1 \quad \forall \mathcal{B} \in ov(\mathcal{I}), \forall t \in \mathcal{T}, \tag{8}$$

$$\sum_{t \in \mathcal{T}_d} e_i^t \le |\mathcal{T}_d| u_i^d \quad \forall i \in \mathcal{I}, \forall d \in \mathcal{D}, \tag{9}$$

$$f_w^t + b_{wi} e_i^t \le 1 \quad \forall w \in \mathcal{W}, \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, \tag{10}$$

$$f_w^d + \widetilde{b}_{wi} \left(u_i^d + u_i^{d+1} \right) \le 2 \quad \forall w \in \mathcal{W}, \forall i \in \mathcal{I}, \forall d \in \mathcal{D}, \tag{11}$$

$$\sum_{t \in \mathcal{T}} e_i^t = \sum_{m \in \mathcal{M}_i} d_{im} x_{im} \quad \forall i \in \mathcal{I}, \tag{12}$$

$$\sum_{r \in \mathcal{R}_i} y_{ri} = \sum_{m \in \mathcal{M}_i} q_{im} x_{im} \quad \forall i \in \mathcal{I}, \tag{13}$$

$$e_i^t + y_{ri} - z_{ri}^t \le 1 \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \forall t \in \mathcal{T}, \tag{14}$$

$$z_{ri}^t \leq y_{ri} \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \forall t \in \mathcal{T}, \tag{15}$$

$$z_{ri}^t \leq e_i^t \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \forall t \in \mathcal{T}, \tag{16}$$

$$\sum_{i \in \mathcal{I}_l \cap \mathcal{R}_i} z_{ri}^t \leq \pi_r^t v_{rl}^t \quad \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{17}$$

$$\sum_{l \in \mathcal{L}} v_{rl}^t = 1 \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \tag{18}$$

$$v_{rl_r^t}^t = 1 \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} s.t. \pi_r^t = 0, \tag{19}$$

$$v_{rl}^t + \sum_{l' \in \mathcal{L} | \sigma_{ll'} = 0} v_{rl'}^{t'} \leq 1 \quad \forall r \in \mathcal{R}, \forall d \in \mathcal{D}, \forall (t, t') \in \mathcal{T}_d \times \mathcal{T}_d, t \neq t', \forall l \in \mathcal{L}, \tag{20}$$

$$e_i^t, s_i^t, c_i^t \in \{0, 1\} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, \tag{21}$$

$$u_i^d \in \{0, 1\} \quad \forall i \in \mathcal{I}, \forall d \in \mathcal{D}, \tag{22}$$

$$f_w^t \in \{0, 1\} \quad \forall w \in \mathcal{W}, \forall t \in \mathcal{T}, \tag{23}$$

$$f_w^d \in \{0, 1\} \quad \forall w \in \mathcal{W}, \forall d \in \mathcal{D}, \tag{24}$$

$$y_{ri} \in \{0, 1\} \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \tag{25}$$

$$z_{ri}^t \in \{0, 1\} \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \forall t \in \mathcal{T}, \tag{26}$$

$$v_{rl}^t \in \{0, 1\} \quad \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}. \tag{27}$$

The objective in (1) is defined as the difference between the profits generated by the turbines and the penalties induced by task outsourcing. Constraints (2) guarantee that exactly one execution mode is selected for each task. For each task, constraints (3)-(6) ensure consistency between the starting time, ending time, and execution time variables. Constraints (7) prevent a task to be scheduled during forbidden periods. Constraints (8) are the non-overlapping constraints. Constraints (9) couple the periods during which each task is performed to the execution days of this task. Constraints (10) and (11) compute the impact of the tasks on the availability of the turbines to produce energy. Constraints (12) connect the duration of each task to its selected execution mode. Constraints (13) ensure that the technician requirements are fulfilled. Constraints (14) force technicians to be assigned to a task for the task's entire duration. For each technician, constraints (15)-(16) ensure consistency between the global assignment and the time-indexed assignment variables. Constraints (17) couple the technicians' locations to the tasks they perform. Constraints (18) prevent technicians to perform multiple tasks during the same period. When technicians are not available, constraints (19) ensure compliance with their known locations. Inequalities (20) are the location-based incompatibility constraints. Finally, (21)-(27) state the binary nature of the decision variables.

2.2. Compact formulation

In order to restrict the number of constraints involved in the formulation $[P1]$, we propose a second model based on the concept of *plans*. A plan associated with task $i \in \mathcal{I}$ defines a feasible schedule for i by setting an execution mode, a consistent starting date, and, by induction, a duration, and a resource requirement. For example, consider a task i^* with two execution modes m_1 and m_2. Let d_{m_1} and d_{m_2} denote the corresponding durations and q_{m_1} and q_{m_2} the corresponding number of required technicians. Assume that i^* can be executed during the whole time horizon. For each $t \in \mathcal{T}$ such as $t \leq |T| - d_{m_1}$, a feasible plan is created to represent the planning of task i^* within mode m_1 from period t to period $t + d_{m_1}$ with a requirement of q_{m_1} technicians. The same procedure is applied for mode m_2.

All the plans are generated a priori. Since in practice the planning horizon is short (because of weather predictions) and there are only a few execution modes, the total number of plans is relatively small. We denote by $\mathcal{P}$ the set of plans, i_p the task involved in plan $p \in \mathcal{P}$, and $\mathcal{P}_i$ the set of all plans involving task i (i.e., $\mathcal{P}_i = \{p \in \mathcal{P} | i_p = i\}$). We also create for each task i a plan $p_i^0 \in \mathcal{P}_i$ representing the outsourcing of the task. For a plan p, execution periods of i_p are expressed by a boolean vector a_p where $a_p^t = 1$ if i_p is executed during period $t \in \mathcal{T}$, and $a_p^t = 0$ otherwise. Similarly, we introduce boolean vector $\widetilde{a}_p$ where $\widetilde{a}_p^d = 1$

if i_p is preempted during the rest period following day $d \in \mathcal{D}$, and $\widetilde{a}_p^d = 0$ otherwise. With a slight abuse of notation, we introduce parameters b_{wp}, $\widetilde{b}_{wp}$, and $\mathcal{R}_p$ respectively equal to b_{wi_p}, $\widetilde{b}_{wi_p}$ and $\mathcal{R}_{i_p}$. Moreover, we define q_p as the number of technicians required when selecting plan $p \in \mathcal{P}$. Finally, parameter o_p is the penalty if plan p is selected (note that $\forall i \in \mathcal{I}, \forall p \in \mathcal{P}_i \setminus \{p_i^0\},\ o_p = 0$ and $o_{p_i^0} = o_i$).

Scheduling the tasks becomes rather implicit as it simply means to select a plan for each task. Nevertheless, we still need to manage the technician-to-task assignment while meeting the incompatibility constraints.

We use the decision variables f_w^t, $\widetilde{f}_w^d$, and v_{rl}^t defined in Section 2.1. We also introduce the following decision variables:

$$\bar{x}_p = \begin{cases} 1 & \text{if plan } p \in \mathcal{P} \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

$$\bar{y}_{rp} = \begin{cases} 1 & \text{if technician } r \in \mathcal{R}_p \text{ is assigned to plan } p \in \mathcal{P}, \\ 0 & \text{otherwise.} \end{cases}$$

As a result, we obtain the following integer linear program, denoted as $[P2]$.

$$[P2] \quad \max \sum_{w \in \mathcal{W}} \left(\sum_{t \in \mathcal{T}} g_w^t f_w^t + \sum_{d \in \mathcal{D}} \widetilde{g}_w^d \widetilde{f}_w^d \right) - \sum_{p \in \mathcal{P}} o_p \bar{x}_p \tag{28}$$

subject to:

$$\sum_{p \in \mathcal{P}_i} \bar{x}_p = 1 \quad \forall i \in \mathcal{I}, \tag{29}$$

$$\sum_{i \in \mathcal{B}} \sum_{p \in \mathcal{P}_i} a_p^t \bar{x}_p \leq 1 \quad \forall \mathcal{B} \in ov(\mathcal{I}), \forall t \in \mathcal{T}, \tag{30}$$

$$f_w^t + b_{wp} a_p^t \bar{x}_p \leq 1 \quad \forall w \in \mathcal{W}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T}, \tag{31}$$

$$\widetilde{f}_w^d + \widetilde{b}_{wp} \widetilde{a}_p^d \bar{x}_p \leq 1 \quad \forall w \in \mathcal{W}, \forall p \in \mathcal{P}, \forall d \in \mathcal{D}, \tag{32}$$

$$\sum_{r \in \mathcal{R}_p} \bar{y}_{rp} = q_p \bar{x}_p \quad \forall p \in \mathcal{P}, \tag{33}$$

$$\sum_{i \in \mathcal{I}_l | r \in \mathcal{R}_i} \sum_{p \in \mathcal{P}_i} a_p^t \bar{y}_{rp} \leq \pi_r^t v_{rl}^t \quad \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{34}$$

$$(18), (19), (20),$$

$$\bar{x}_p \in \{0,1\} \quad \forall p \in \mathcal{P}, \tag{35}$$

$$\bar{y}_{rp} \in \{0,1\} \quad \forall p \in \mathcal{P}, \forall r \in \mathcal{R}_p, \tag{36}$$

$$(23), (24), (27).$$

The objective in (28) is defined as the difference between the profits generated by the turbines and the penalties induced by task outsourcing. Constraints (29) ensure that exactly one plan is selected for each task. Constraints (30) are the non-overlapping constraints. Constraints (31) and (32) couple energy production variables to plan selection variables. Constraints (33) ensure that the technician requirements are fulfilled. Constraints (34) couple the technicians location to the tasks they perform. This new formulation $[P2]$ uses the same constraints as formulation $[P1]$ to manage the technicians locations. Finally, (35)-(36) state the binary nature of the new decision variables.

To potentially improve the quality of the linear relaxation upper bound, and consequently speeding up the solution, we add a disaggregated form of constraints (33) that prevent a technician to be assigned to a plan p if the plan is not selected.

$$\forall p \in \mathcal{P}, \forall r \in \mathcal{R}_p, \quad \bar{y}_{rp} \leq \bar{x}_p \tag{37}$$

3. Constraint programming formulation

Alternatively, the problem can be approached using constraint programming. The CP paradigm is a powerful and flexible tool which makes expressing complex constraints relatively easy. It has been successfully applied for a variety of scheduling problems (Baptiste et al., 2001; Rodriguez, 2007; Malapert et al., 2012). In the remainder of this section, we introduce a CP model for our problem.

First of all, note that selecting: i) an execution mode, ii) a starting time, and, iii) a technician assignment for each task, is enough to obtain a solution to our problem. Therefore, for each task $i \in \mathcal{I}$, we denote $M_i \in \mathcal{M}_i$ its execution mode, $S_i \in \mathcal{T}$ its starting period, and we use binary variables $(y_{ri})_{r\in\mathcal{R}_i}$ introduced in Section 2.1 to decide if technician r performs or not task i. To make some constraints easier to model, we introduce integer variables $C_i \in \mathcal{T}$, $D_i \in \{d_{im}\}_{m\in\mathcal{M}_i}$, $Q_i \in \{q_{im}\}_{m\in\mathcal{M}_i}$ and set variables $E_i \subseteq \mathcal{T}$ defining for task i its ending time, its duration, its number of assigned technicians, and its set of execution periods, respectively.

Execution periods of each task are coupled to their starting and ending time with constraints (38)-(39).

$$S_i + D_i = C_i \quad \forall i \in \mathcal{I}, \tag{38}$$
$$t \in E_i \Leftrightarrow t \in [S_i; C_i[\cap \mathbb{N} \quad \forall i \in \mathcal{I} \tag{39}$$

The duration of each task (40) as well as the number of assigned technicians (41) are coupled with the selected execution mode.

$$D_i = d_{iM_i} \quad \forall i \in \mathcal{I}, \tag{40}$$
$$Q_i = q_{iM_i} \quad \forall i \in \mathcal{I} \tag{41}$$

Constraints (42) are the non-overlapping constraints.

$$\bigcap_{i\in\mathcal{B}} E_i = \emptyset \quad \forall \mathcal{B} \in ov(\mathcal{I}) \tag{42}$$

Constraints (43) ensure that the technician requirements are fulfilled for each task.

$$\sum_{r\in\mathcal{R}_i} y_{ri} = Q_i \quad \forall i \in \mathcal{I} \tag{43}$$

To forbid a technician $r \in \mathcal{R}$ to be assigned to multiple tasks during a period, we introduce set variables $Y_r^t \subseteq \mathcal{I} \cup \{i^0\}$ defining the set of tasks that the technician could potentially perform at period $t \in \mathcal{T}$. Index i^0 represents a dummy task, created in order to prevent a technician to work during unavailability periods. Constraints (44) couple these variables to the global assignment variables $(y_{ri})_{i\in\mathcal{I}|r\in\mathcal{R}_i}$. Restrictions imposed on the locations visited by a technician within each day lead to the introduction of set variables $V_r^t \subseteq \mathcal{L}$ defining the set of potential locations for technician r during period t. Constraints (45)-(46) restrict the set of tasks that a technician can possibly execute according to his or her potential locations. Set $\mathcal{L}(\hat{\mathcal{I}})$ defines the set of locations where the tasks in set $\hat{\mathcal{I}}$ have to be performed. Note that $\mathcal{L}(\hat{\mathcal{I}}) = \{l \in \mathcal{L} \mid \exists i \in \hat{\mathcal{I}} \ s.t.\ l_i = l\}$.

$$y_{ri} = 1 \Rightarrow \left(Y_r^t = \{i\} \quad \forall t \in E_i\right) \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}_i, \tag{44}$$
$$V_r^t = \mathcal{L}(Y_r^t) \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \ s.t.\ \pi_r^t = 1, \tag{45}$$
$$V_r^t = \{l_r^t\} \wedge Y_r^t = \{i^0\} \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \ s.t.\ \pi_r^t = 0 \tag{46}$$

Constraints (47) ensure that technicians satisfy every day the location-based incompatibility constraints.

$$V_r^t = \{l\} \Rightarrow \Big(l' \notin V_r^{t'} \quad \forall l' \in \mathcal{L},\ s.t.\ \sigma_{ll'} = 0 \Big) \quad \forall r \in \mathcal{R}, \forall d \in \mathcal{D}, \forall t, t' \in \mathcal{T}_d\ s.t.\ t' \neq t, \forall l \in \mathcal{L} \tag{47}$$

In order to define the objective function of our problem, we introduce two set variables. Variables $F_w^{day} \subseteq \{1, \ldots, |\mathcal{T}|\}$ define the set of all periods during which turbine $w \in \mathcal{W}$ can produce energy. Variables $F_w^{rest} \subseteq \{1, \ldots, |\mathcal{D}|\}$ define the set of days for which turbine w can produce energy during the corresponding rest periods. More specifically, a day d belongs to this set if w can produce energy during the rest period between d and $d + 1$. Additionally, we denote by t_d^{rest} the last period $t \in \mathcal{T}$ before the rest period following day $d \in \mathcal{D}$.

We therefore introduce the following constraints:

$$t \in E_i \Rightarrow t \notin F_w^{day} \quad \forall w \in \mathcal{W}, \forall i \in \mathcal{I}\ s.t.\ b_{wi} = 1, \forall t \in \mathcal{T}, \tag{48}$$

$$t \notin \bigcup_{i \in \mathcal{I} | b_{wi} = 1} E_i \Rightarrow t \in F_w^{day} \quad \forall w \in \mathcal{W}, \forall t \in \mathcal{T}, \tag{49}$$

$$t_d^{rest} \in E_i \wedge (t_d^{rest} + 1) \in E_i \Rightarrow d \notin F_w^{rest} \quad \forall w \in \mathcal{W}, \forall i \in \mathcal{I},\ s.t.\ \widetilde{b}_{wi} = 1, \forall d \in \mathcal{D}, \tag{50}$$

$$\bigwedge_{i \in \mathcal{I} | \widetilde{b}_{wi} = 1} \big(\{t_d^{rest}, (t_d^{rest} + 1)\} \notin E_i \big) \Rightarrow d \in F_w^{rest} \quad \forall w \in \mathcal{W}, \forall d \in \mathcal{D} \tag{51}$$

Constraint (52) defines the objective function variable $obj \in \mathbb{N}$ of our problem.

$$obj = \sum_{w \in \mathcal{W}} \left(\sum_{t \in F_w^{day}} g_w^t + \sum_{d \in F_w^{rest}} \widetilde{g}_w^d \right) - \sum_{i \in \mathcal{I} | M_i = m_i^0} o_i \tag{52}$$

To remove some symmetries, we add constraints (53) to impose the starting time of an outsourced task to be equal to 0.

$$M_i = m_i^0 \Leftrightarrow S_i = 0 \qquad \forall i \in \mathcal{I} \tag{53}$$

4. A CP-based large neighborhood approach

We use the CP model introduced in Section 3 as the main building block of a CP-based large neighborhood search (CPLNS) approach.

This method is based on the *large neighborhood search* metaheuristic (LNS) originally proposed by Shaw (1998) for a vehicle routing problem. In the LNS, the current solution is successively destroyed and repaired in order to improve its quality. The adaptive large neighborhood search (ALNS), introduced by Ropke and Pisinger (2006), is an extension of the LNS applying several destroy and repair operators. In ALNS, operators are selected with an adaptive layer based on their performance in previous iterations. After some preliminary experimentation we decided to drop the adaptive layer because its contribution to the accuracy of the method did not payoff the loss of simplicity and the effort needed to fine tune the additional parameters. We then randomly select operators with equal probability as suggested in (Pisinger and Ropke, 2010).

Algorithm 1 presents the general structure of the method. The algorithm involves an iterative process. In every iteration, a destroy operator o_1 and a repair operator o_2 are randomly selected. First, the current solution is destroyed using o_1 (see Section 4.1). Then, a new solution sol' is built using the repair operator o_2 (see Section 4.2). In the case where the resulting solution sol' meets the acceptance criteria (see Section 4.3), it replaces the current solution sol for the next iteration. If sol' is better than the best solution found so far, it replaces $sol*$. Then, the search moves to the next iteration. The whole procedure is repeated until one of the stopping criteria is met.

Algorithm 1: Script of the *CPLNS* algorithm

```
1  $sol \leftarrow$ initial solution
2  $sol^* \leftarrow sol$
3  repeat
4      Select a destroy operator $o_1$ and a repair operator $o_2$ from the operators pool
5      $sol^{'} \leftarrow$ repair $(o_2, \textbf{destroy}(o_1, s), sol)$
6      if $sol^{'}$ is accepted then
7          $sol \leftarrow sol^{'}$
8      end
9      if $f(sol^{'}) > f(sol^*)$ then
10         $sol^* \leftarrow sol^{'}$
11     end
12 until one of the stopping criteria is met;
13 return $sol^*$
```

4.1. Destroy operators

At each iteration, the algorithm selects Γ tasks to remove from the current solution. The value of Γ is randomly fixed in the interval $[\max(n^-, n \times p^-); \min(n^+, n \times p^+)]$, where n^- and n^+ denote the minimal and maximal number of tasks that are allowed to be removed during an iteration; similarly, p^- and p^+ denote the minimal and maximal proportion of tasks that could be removed. The parameters p^- and p^+ allow the algorithm to adapt to all instances independently of their size. We use the following settings: $(n^-, n^+, p^-, p^+) = (5, 20, 0.1, 0.4)$. We also always consider outsourced tasks in the current solution as tasks to be removed. However, we do not count them among the Γ tasks to remove.

After setting Γ, the algorithm selects the tasks using one of the following six removal operators:

- Operator A: random removal
  This operator randomly removes Γ tasks from the current solution. The intuition behind this operator is to diversify the search.

- Operator B: worst removal
  This operator removes the tasks which penalize the most the objective function of the current solution. Let f be the current value of the objective function, f_{-i} its value if task i is not considered anymore, and $\Delta f(i) = f - f_{-i}$. The tasks with the greatest values of $\Delta f(i)$ are removed from the current solution in order to insert them at better positions.

- Operator C: technicians duties removal
  This operator randomly selects a skill s^*. Then it randomly selects technicians mastering s^*. Next, it removes from the current solution those tasks in which the selected technicians use s^*. The procedure is repeated until Γ tasks are removed. Freeing up a pool of technicians along the whole time horizon may allow the reinsertion of possibly misplaced tasks in more convenient periods (i.e, periods where they penalize less the profits).

- Operator D: similar tasks removal
  This operator removes *similar tasks.* More specifically, the operator removes non-overlapping tasks (or tasks that overlap as little as possible) having similar duration and skill requirements. The similarity $\phi(i, j)$ between two tasks $i, j \in \mathcal{I}$ in a solution *sol* is formally defined as: $\phi(i, j, sol) = \alpha_1 \times |\bar{d}_i - \bar{d}_j| + \alpha_2 \times \mathbb{1}_{(s_i \neq s_j)} + \alpha_3 \times ov(i, j, sol)$, where $\bar{d}_i$ is the average duration of task i (i.e., $\bar{d}_i = \frac{1}{|\mathcal{M}_i|} \sum_{m \in \mathcal{M}_i} d_{im}$). Function $ov(i, j, sol)$ computes the number of overlapping periods between i and j in the current solution *sol*. Coefficients α_1, α_2, α_3 weight the three components of the similarity function, namely, the

tasks duration, the skills requirements, and the tasks overlapping. In our experiments, $(\alpha_1, \alpha_2, \alpha_3) = (1, 3, 5)$. To select the tasks to remove, the operator first initializes a set $\widetilde{\mathcal{I}}$ with a random task. While $|\widetilde{\mathcal{I}}| \leq \Gamma$, the procedure randomly selects a task i^* from $\widetilde{\mathcal{I}}$ and then it adds to $\widetilde{\mathcal{I}}$ the task $j \in \mathcal{I} \setminus \widetilde{\mathcal{I}}$ with the minimal value of $\phi(i^*, j)$. The intuition behind this operator is to remove similar tasks that are scheduled in non-overlapping periods in order to increase the likelihood of a solution improvement.

- Operator E: turbine maximal regret
  This operator removes the tasks having the largest difference between the current loss of profits they generate and the minimal loss of profits they can induce. For each turbine $w \in \mathcal{W}$, the procedure first computes a metric called $loss_w^{best}$ estimating the smallest profit loss that can be achieved when only the tasks having an impact on I_w (i.e., the tasks that prevent turbine w to produce energy when scheduled) are considered. The value of $loss_w^{best}$ is computed using the CP formulation of Section 3 on the underlying instance. Let $\mathcal{I}_w$ be the tasks having an impact on the energy production of turbine w. Let $\overline{loss}_w$ be the profit loss generated in the current solution sol by the tasks in $\mathcal{I}_w$. The costs related to outsourced tasks are included in the computation of $\overline{loss}_w$. Let w^* denote the turbine associated to the largest value of $\Delta loss(w) = loss_w^{best} - \overline{loss}_w$. The procedure removes all the tasks from $\mathcal{I}_{w^*}$ from the current solution sol. Removing tasks that currently generate considerably more lost profit than they could may allow the algorithm to schedule those tasks in better positions in the next iteration. It is then plausible to assume that this operator increase the probability to find better quality solutions.

- Operator F: task maximal regret
  This operator works almost in the same way as operator E. Instead of reasoning by turbine, we focus on the loss of profits induced by each scheduled task. Let $\mathcal{W}_i$ denote the set of turbines shutdown by the execution of a task i. Clearly, $\mathcal{W}_i = \{w \in \mathcal{W} | b_{wi} = 1 \vee \widetilde{b}_{wi} = 1\}$. The loss induced by i is equal to the sum over all the turbines in $\mathcal{W}_i$ of the profit lost directly due to the scheduling of i. If multiple tasks impact a turbine at a specific period, the loss is set proportionally to the number of these tasks.

We work with randomized versions of operators B, D, E and F to explore the search space more broadly. Indeed, an operator can destroy different parts of the same solution each time it is applied on it. This can then lead to build different solutions. Although the randomization strategy we use is relatively simple, we explain it here for the sake of completeness. The strategy is based on the one proposed in (Cordeau et al., 2010). Let ϱ_o denote the *randomization factor* of operator o. When selecting tasks for removal, the operator first sorts a list L containing all the tasks using its selection criterion (e.g., largest penalization for operator B, largest similarity with a specified task for operator D, largest regret for operators E and F). The first positions of L contains the tasks that the destroy operator has to target first according to its criterion. Then the operator draws a random number $y \in [0; 1[$ and it selects for removal task i in position $\lfloor y^{\varrho_o} \times |L| \rfloor$ in L (positions in L are indexed from 0). A randomization factor $\varrho_o = 1$ makes the operator completely random, while higher values of ϱ_o make the operators more deterministic. In our experiments we set $\rho_B = \rho_D = \rho_E = \rho_F = 3$ and we use only the randomized versions of these four operators.

Although it is very simple, Algorithm 2 presents the general structure of a destroy operator used as a subroutine in Algorithm 1.

Algorithm 2: Destroy(o,sol)

```
   Data: a solution $sol$
   a destroy operator $o$
   Result: a set of tasks to remove from $sol$
1  $\mathcal{F} \leftarrow \emptyset$
2  $\mathcal{F} \leftarrow$Apply destroy operator $o$ to $sol$
3  return $\mathcal{F}$
```

4.2. Repair operators

We use the CP formulation introduced in Section 3 to repair partially destroyed solutions. More specifically, if $\mathcal{F}$ denotes the set of tasks that have been removed, we fix the values of the variables M_i, S_i and $(y_{ri})_{r\in\mathcal{R}_i}$ for all tasks $i \in \mathcal{I} \setminus \mathcal{F}$ and solve the resulting model.

A solution to the CP model is found as soon as the decision variables (M_i, S_i and $(y_{ri})_{r\in\mathcal{R}_i}$) are instantiated for all task $i \in \mathcal{I}$. Therefore, the branching strategy should focus only on these variables. It is worth noting that a CP solver can only make deductions when executing modes and starting times variables are instantiated for the tasks. Indeed, the duration and the technician requirements of each task are only defined when the execution mode is fixed. Then, fixing the starting time of a task before knowing its execution mode leads to a weak propagation. Moreover, variables y_{ri} have an impact only on the feasibility of a solution but not on its quality. Fixing last these variables (i.e., after having fixed the variables M_i and S_i for all tasks $i \in \mathcal{I}$) implies that the solver has to explore a large sub-tree before decision refutations that may lead to an improvement of the best solution. Based on these observations we adopt a task-by-task scheduling strategy in which the technicians assignment is made after having chosen an execution mode and a starting time for the current task.

It is well-known that quickly reaching a good-quality solution increases the efficiency of the search. It is, however, not clear if fixing the execution mode of a $i \in \mathcal{I}$, M_i, to a specific execution mode and then exploring all its potential starting times before setting M_i to another value is the best searching strategy. This observation leads to set simultaneously variables M_i and S_i to achieve a greater flexibility during the search. We choose therefore to reuse the notion of plan introduced in Section 2.2. For each task $i \in \mathcal{I}$, we introduce variable $X_i \in \mathcal{P}_i$ that defines the plan selected for task i. We add the constraints (54)-(55) to couple these variables to the variables M_i and S_i.

$$M_i = mode_{X_i} \quad \forall i \in \mathcal{I}, \tag{54}$$

$$S_i = start_{X_i} \quad \forall i \in \mathcal{I} \tag{55}$$

For a plan $p \in \mathcal{P}$, $mode_p$ is the selected execution mode for the task i_p and $start_p = \min_{t\in\mathcal{T}|a_p^t=1} t$ represents the starting time of i_p. In summary, task by task, we first define its execution mode along with its starting time by fixing variable X_i, and we finally assign the required technicians by fixing the variables $(y_{ri})_{r\in\mathcal{R}_i}$.

To reach feasible solutions faster, we design custom propagators that try to keep the domain of X_i consistent during the search. These propagators rely on the number of technicians available at each period of the time horizon considering the required skills and the locations incompatibility constraints. They also take into account that technicians have to work on tasks from the beginning to the end.

The most critical part of the procedure is the selection of the next task to be considered by the branching strategy. We select the next task to schedule using a look-ahead regret heuristic. This heuristic tries to anticipate future decisions. We then select the task with the largest regret value. The regret value of a task is an estimation, according to the current partial schedule, of the additional cost incurred if the task is not scheduled by selecting the plan leading to the smallest loss of profits.

Let $\mathcal{I}^0$ denote the set of tasks which have not yet been processed at the current node of the search. We denote Δf_i^k the k-th smallest value of the loss of profits that task i can generate when scheduled using one of its possible plans. The procedure *regret-q* chooses task $i^* = \arg\max_{i\in\mathcal{I}^0} \sum_{k=2}^{k=q} \left(\Delta f_i^k - \Delta f_i^1\right)$ to be considered for scheduling. The algorithm computes Δf_i^k according to the values of $\Psi(i,p)$, a function representing the loss of profit if task i uses plan $p \in \mathcal{P}_i$ (i.e., performed in mode $mode_p$ starting at period $start_p$). Function $\Psi(i,p)$ is computed using functions $\Psi^{day}(i,p)$ and $\Psi^{rest}(i,p)$ which represent the loss of profits during the periods from $\mathcal{T}$ and during the rest periods if task i uses plan $p \in \mathcal{P}_i$. These functions are defined as follows:

$$\Psi(i,p) = \Psi^{day}(i,p) + \Psi^{rest}(i,p),$$

$$\Psi^{day}(i,p) = \begin{cases} o_p & \text{if } p = p_i^0, \\ \sum\limits_{w \in \mathcal{W} | b_{wi}=1} \sum\limits_{t=start_p}^{t<start_p+d_{im}} g(w,t) & \text{otherwise.} \end{cases},$$

$$\Psi^{rest}(i,p) = \begin{cases} 0 & \text{if } p = p_i^0, \\ \sum\limits_{w \in \mathcal{W} | \widetilde{b}_{wi}=1} \sum\limits_{d \in \bar{\mathcal{D}}_p} \widetilde{g}(w,d) & \text{otherwise.} \end{cases},$$

where $\bar{\mathcal{D}}_p = \{d \in D | start_p \leq t_d^- \wedge t_d^+ < start_p + d_{im_p}\}$. Functions $g(w,t)$ and $\widetilde{g}(w,d)$ are defined as:

$$\forall w \in \mathcal{W}, \forall t \in \mathcal{T}, \quad g(w,t) = \begin{cases} g_w^t & \text{if } t \in Env(F_w^{day}), \\ 0 & \text{otherwise.} \end{cases},$$

$$\forall w \in \mathcal{W}, \forall d \in \mathcal{D}, \quad \widetilde{g}(w,d) = \begin{cases} \widetilde{g}_w^d & \text{if } d \in Env(F_w^{rest}), \\ 0 & \text{otherwise.} \end{cases},$$

where $Env(X)$ denotes the set of elements that may belong to the set variable X in a solution at the current node of the search tree.

Let $Dom(x)$ denote the domain of variable x (i.e., all the possible values that x can take). We have $\Delta f_i^1 = \min\limits_{p \in Dom(X_i)} \Psi(i,p)$. More generally, Δf_i^k is the k-th smallest value of $\Psi(i,p)$. Once task i^* has been selected, it is scheduled using plan $p^* = \arg\min\limits_{p \in Dom(X_i)} \Psi(i,p)$.

Sometimes, we observe that the look-ahead heuristic can be inadequate to obtain good solutions. It is indeed possible that a task with a small regret at a given point of the search is not chosen to be scheduled but that this decision leads to a large loss of profits in the associated subtree. To overcome this possible issue, we design another branching strategy that selects the task $i^* = \arg\max\limits_{i \in \mathcal{I}_0} \left(\min\limits_{p \in \mathcal{P}_i} \Psi_(i,p) \right)$ for which the minimal loss of profits is maximal. We refer to this branching strategy as $MinMaxLoss$.

The resources assignment is then done technician by technician as long as the request is not fulfilled. Let $\mathcal{D}_p$ be the set of days that task i_p of plan $p \in \mathcal{P}$ overlaps. We choose as a priority the compatible technician which is already working during the days that belong to $\mathcal{D}_{p^*}$. Since incompatibility constraints (47) are very restrictive, it should be preferable to use technicians that are already working at the same location or at compatible locations. Otherwise, it could restrict drastically the number of technicians that will be available for other tasks, especially those at incompatible locations. If during the days $d \in \mathcal{D}_{p^*}$ multiple technicians work the same number of periods, we choose first the technician that could perform the least number of tasks among those remaining. If several technicians can still be selected, we select one randomly.

Exploring the whole neighborhood of a solution is time-consuming; therefore we only allow a certain number ϖ_{max} of backtracks (we set $\varpi_{max} = 200$ in our experiments). Thus, different solutions can be obtained using different branching strategies. Different repair operators are therefore defined using different branching strategies. In our experiments, we use regret-2 and regret-3 branching strategies, as well as a randomized version of $MinMaxLoss$, where the probability of selecting a task is inversely proportional to the minimal loss of profits it generate at this point of the search.

Algorithm 3 presents the general structure of a repair operator used as a subroutine in Algorithm 1.

4.3. Acceptance criteria

The original version of LNS proposed by Shaw (1998), uses an elitist strategy to accept solutions (i.e., it accepts only improving solutions). On the other hand, the LNS by Pisinger and Ropke (2007) uses the Metropolis criterion to accept solutions. According to this criterion, solutions are accepted with a given probability. If the newly found solution s' improves the current solution s the probability equals one. Otherwise, the probability is computed using the Boltzmann expression: $e^{-\left(f(s)-f(s')\right)/\Upsilon}$. Parameter Υ

Algorithm 3: Repair(o,$\mathcal{F}$,s)

```
   Data: a solution $sol$
   a set $\mathcal{F}$ of tasks
   a repair operator $o$ (branching strategy)
   Result: a new solution $sol'$
1  foreach $i \in \mathcal{I} \setminus \mathcal{F}$ do
2      Fix the values of $M_i$, $S_i$, $(y_{ri})_{r \in \mathcal{R}_i}$ as in solution $sol$ in the CP model
3  end
4  Solve the CP model applying repair operator $o$, yelding $sol'$
5  return $sol'$
```

is refereed as the temperature. It is updated after each iteration using what is commonly known as the geometric cooling schedule: $\Upsilon = \Upsilon \times \bar{c}$, where $\bar{c} \in [0, 1[$. Then, the probability for accepting non-improving solutions decreases over the iterations. We tested the two approaches in our experiments.

We also tested a mix of them: we apply an elitist strategy during the k first iterations, and then we activate the Metropolis criterion. We based our choice in two observations. First, using the elitist strategy, the search is often trapped in local optima after a certain amount of iterations, and then it struggles to improve the solution. Second, as we do not ensure that our algorithm starts from a good-quality solution, reaching a good solution can be time-consuming.

In our experiments, k is set to $1,000$ and $\bar{c}$ to 0.99975. The initial temperature is fixed to $-\frac{0.25}{ln 0.5} f(s_0)$ where $f(s_0)$ is the value of the objective function of the initial solution s_0. Therefore, in the first iteration our approach accepts solutions that are 2.5% worse than the current solution with a probability of 0.5.

5. Computational experiments

We implemented our algorithms using *Java 8 (JVM 1.8.0.25)*. We relay on *Gurobi 6.0.4* for solving the IP models [$P1$] and [$P2$] and *Choco 3.3.1* for solving the CP formulation (see Prud'homme et al. (2014)). We ran our experiments on a Linux 64 bit-machine, with an Intel(R) Xeon(R) X5675 (3.07Ghz) and 10GB of RAM.

5.1. Instance families

To test our approach, we generated a set of instances based on a real-world application. Our testbed is composed of 60 instances. For each instance, we consider different time horizons of different lengths (20 or 40), number of tasks (30, 60 or 90), and number of skills (1 or 3). Each task can be executed in several modes (1 to 4). Note that $|\mathcal{S}| = 1$ simply means that no skills are considered. We generate the profit values following three steps. We first generate wind speed values at each location. We then compute the production generated by each turbine based on the work of Giorsetto and Utsurogi (1983). Finally, we fix the profit values using a wind energy market price. We did not consider the alternative objective (i.e., maximizing the availability of the turbines) as the problem becomes much more easier since the availability of the turbines is essentially impacted by the execution mode chosen for each task but not by their starting time. For a thorough discussion on the instance generation process and a full description of our instances the reader is referred to Appendix Appendix B. Table 1 presents the twelve families of instances in our testbed.

In order to assess the quality of the results, we compute the gap to the optimal solution when it is known or, otherwise, to the minimal upper bound found by Gurobi when solving the two integer linear formulations after 3 hours of branch-and-bound.

5.2. IP formulations

Table 2 shows the average number of variables and constraints introduced by the two integer linear formulations [$P1$] and [$P2$] for each family of instances. Formulation [$P2$] contains in average 1.5 more

Table 1: Instance families

Family	$\|\mathcal{T}\|$	$\|\mathcal{I}\|$	$\|\mathcal{S}\|$	# instances
20_30_1	20	30	1	5
20_30_3	20	30	3	5
20_60_1	20	60	1	5
20_60_3	20	60	3	5
20_90_1	20	90	1	5
20_90_3	20	90	3	5
40_30_1	40	30	1	5
40_30_3	40	30	3	5
40_60_1	40	60	1	5
40_60_3	40	60	3	5
40_90_1	40	90	1	5
40_90_3	40	90	3	5

variables than formulation $[P1]$ but it divides by more than 4 the number of constraints to consider. This impacts significantly the results obtained when solving the models.

These results are summarized in Table 3. For each formulation, we report the average (Avg), minimal (Min) and maximal (Max) gap obtained for each family of instances. We also report the number of optimal solutions found within the time limit (# Opt), the average resolution time (Time), and a ratio between the upper bound computed by the formulation and the minimal upper bound computed by the two formulations (UB ratio). We observe that formulation $[P2]$ outperforms formulation $[P1]$. We believe these results can be explained as follows. First, the compact formulation contains far less constraints than the natural formulation. Second, the upper bound computed using formulation $[P2]$ is always tighter and in average 1.3% tighter than the one computed using the formulation $[P1]$. On the two families of larger instances, formulation $[P2]$ provides worst results than formulation $[P1]$ but these results should be interpreted with care since results are still very far from the optimal solution. As a matter of fact, optimality is proved only for few instances within the 3-hour time limit.

In conclusion, solving directly the IP formulations does not seem an efficient solution method as the average gap is still considerable for several family of instances.

Table 2: Average number of variables and constraints introduced by the two integer linear formulations

Family	**[P1]**		**[P2]**	
	#Vars	**#Cts**	**#Vars**	**#Cts**
20_30_1	12,397	28,835	16,364	7,163
20_30_3	12,099	26,986	14,678	6,537
20_60_1	41,788	103,762	61,453	17,997
20_60_3	37,002	89,094	47,775	14,668
20_90_1	85,870	217,641	135,168	32,217
20_90_3	78,276	194,699	102,786	27,787
40_30_1	19,343	47,698	28,156	20 134
40_30_3	20,449	49,678	26,029	18,620
40_60_1	63,610	167,247	96,763	47,791
40_60_3	62,130	161,173	77,028	43,044
40_90_1	140,938	381,217	224,029	87,299
40_90_3	127,415	339,880	152,922	72,575
All	**58,443**	**150,659**	**81,929**	**32,986**

Table 3: Computational results for the two integer linear formulations

Family	**Gap**						**#Opt**		**Time (s)**		**UB ratio**	
	[P1]			**[P2]**			**[P1]**	**[P2]**	**[P1]**	**[P2]**	**[P1]**	**[P2]**
	Avg	Min	Max	Avg	Min	Max						
20_30_1	0.9%	0.0%	3,2%	0.0%	0.0%	0.0%	0	5	10,800	1,838	1.003	1.000
20_30_3	6.3%	0.0%	24.5%	0,2%	0.0%	1.0%	1	2	9,024	7,037	1.016	1.000
20_60_1	2.8%	0.0%	12.0%	0.0%	0.0%	0.0%	0	5	10,800	1,025	1.001	1.000
20_60_3	156.9%	3,2%	524,2%	2.4%	0.7%	4.3%	0	0	10,800	10,800	1.013	1.000
20_90_1	7.6%	0.4%	15.6%	0.0%	0.0%	0.0%	0	5	10,800	2,135	1.002	1.000
20_90_3	19.7%	10.4%	28,2%	3.0%	1.9%	4,2%	0	0	10,800	10 800	1.011	1.000
40_30_1	8.4%	0.7%	18.8%	5.0%	0.0%	12,2%	0	1	10,800	10 790	1.008	1.000
40_30_3	52.8%	3.1%	224.1%	10.3%	2.7%	19.1%	0	0	10,800	10,800	1.017	1.000
40_60_1	95.0%	33.8%	247.9%	64.4%	6.0%	260.0%	0	0	10,800	10,800	1.006	1.000
40_60_3	185.4%	16,2%	269.9%	138.7%	24.0%	340.4%	0	0	10,800	10,800	1.040	1.000
40_90_1	104.3%	69.3%	158.7%	166.4%	16.9%	228.4%	0	0	10,800	10,800	1.003	1.000
40_90_3	63.6%	30.6%	101.7%	149.4%	10.6%	457.4%	0	0	10,800	10,800	1.033	1.000
All	**58.6%**	**0.0%**	**524,2%**	**45.0%**	**0.0%**	**457.4%**	**1**	**18**	**10,651**	**8,202**	**1.013**	**1.000**

5.3. *CP formulation*

Table 4 summarizes the global results found solving directly the CP model with the default branching strategy of the solver (BS1), with our regret-2 branching strategy (BS2), and with a randomized version of the latter coupled to a geometrical restart policy based on the number of backtracks (BS3). On one hand, the default strategy struggles when it comes to finding feasible solutions. This is somehow expected since we work with many different kinds of variables. On the other hand, a solution is always quickly found using our regret branching strategy. Guaranteeing feasibility is, however, relatively easy since tasks can be outsourced.

Additionally, we observe that the initial solutions are little improved during the search when using the regret-2 branching. It seems that the CP model is facing some symmetry issues. One way to overcome this drawback is to restart frequently the search from the root node. Jointly using a randomized branching strategy allows us to explore different parts of the search tree which increases the likelihood of finding better solutions. The results confirm this intuition as the average gap is improved approximately by 10%.

Table 4: Aggregated computational results solving the CP model using different branching strategies

	# instances with a feasible solution		
	1 min	**5 min**	**30 min**
BS1	3	4	4
BS2	60	60	60
BS3	60	60	60

	Average gap		
	1 min	**5 min**	**30 min**
BS1	1144.8%*	989.4%*	985.3%*
BS2	20.7%	19.3%	19.1%
BS3	11.1%	10.0%	9.4%

*: Computed for instances with a feasible solution

5.4. *CPLNS*

Previous results show the limitation of solving directly the CP model. This demonstrates the relevance of a CPLNS approach. For this latter method, we imposed a time limit as the stopping criterion. Since the neighborhoods are partially randomized, we launched the algorithm ten times for each instance.

We first analyze the relevance of combining an elitist strategy with the Metropolis acceptance criterion. Table 5 summarizes the results found with different time limits: 1,5 or 30 minutes; and different solution acceptance criteria: elitism (El), Metropolis (MT), and both (El+MT). Coupling an elitist strategy with the Metropolis acceptance criterion proves to lead to the best results independently of the time limit. We therefore use this acceptance criterion in the remainder of our experiments.

Table 5: Computational results according to the solution acceptance criterion

Time limit	**Average gap**		
	1 min	**5 min**	**30 min**
El	4.9%	3.6%	2.7%
MT	5.9%	4.5%	2.5%
El + MT	4.9%	3.6%	2.5%

We now discuss more thoroughly the performance of the CPLNS algorithm. Table 6 reports the performance of the CPLNS for each instance family. It shows in the different columns the relative average mean gap [3](Mean), the average best gap[4](Best), and the average worst gap[5](Worst) for the CPLNS with 1 minute, 5 minutes, and 30 minutes of time limit. Since optimality is proved only for 40% of the instances, assessing the intrinsic quality of the CPLNS using only the gap is not conclusive enough. However, the overall average gaps of 4.9%, 3.6% and 2.5% after 1 minute, 5 minutes and 30 minutes of CPU time, respectively, show the effectiveness of our approach. We also observe that the algorithm provides near optimal solutions for instances with a 20-length time horizon and no skills. However, the CPLNS is less effective on the instances with a 40-length time horizon, 3 skills, and more than 30 tasks. It is worth noting that the difficulty of the instances seems to be related to the length of the time horizon and the number of skills. Conversely, the number of tasks seems to only have a relatively small impact. Last but not least, our method seems to have a suitable stability as in average the difference between the best and the worst solution found over the 10 runs is a reduced 1.3%.

6. Conclusions and research perspectives

In this study, we have introduced a new and challenging maintenance scheduling problem faced by the wind power industry. Some of the special features of this problem are the presence of alternative execution modes for each task and an individual management of the technicians through a space-time tracking. We have also introduced an original objective function, far from the classical scheduling concerns, linking the profits to the periods during which the maintenance operations are performed. We have conducted computational experiments on instances with up to 90 tasks, 4 execution modes, and 3 skills.

We have proposed three mathematical formulations based on both integer and constraint programming. Computational results indicate that, generally, the models cannot be directly used to solve realistic instances. IP models are unable to solve to optimality most of the instances after 3 hours and the gap is still very large for many families of instances. Nevertheless, we have shown that a IP compact formulation using the notion of plans outperforms the more natural IP formulation of the problem. The number of skills and the length of the time horizon appear to make the problem difficult to solve as their values increase. Similarly, the performance of our CP model seems to be affected by symmetry issues.

To provide an alternative solution approach, we have developed a CP-based large neighborhood search. We have successfully adapted some destroy operators to this new problem and have proposed some new ones. Moreover, we have designed several branching strategies to effectively repair solutions solving a CP model with fixed variables. We have also introduced and have demonstrated the relevance of a new acceptance criterion combining elitism and Metropolis. The CPLNS shows an average gap of 2.5% with respect to

[3]average of the mean gap found for each instance over 10 runs
[4]average of the best (minimal) gap found for each instance over 10 runs
[5]average of the worst (maximal) gap found for each instance over 10 runs

Table 6: Computational results for the CPLNS (average over 10 runs)

Family	Average gap								
	1 min			5 min			30 min		
	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst
20_30_1	0.4%	0.3%	0.6%	0,2%	0.1%	0.3%	0.1%	0.1%	0.3%
20_30_3	1.7%	0.9%	2.7%	0.9%	0.4%	1.8%	0.8%	0,2%	1.6%
20_60_1	0.5%	0.3%	0.7%	0.5%	0.3%	0.6%	0.1%	0.0%	0,2%
20_60_3	3.9%	3.0%	5.0%	3.5%	2.7%	4,2%	1.6%	1.1%	2.1%
20_90_1	0.7%	0.4%	1.1%	0.6%	0.3%	0.8%	0,2%	0.1%	0.4%
20_90_3	4.9%	3.8%	6.3%	4.1%	3,2%	5.0%	2.7%	2.3%	3.3%
40_30_1	2.3%	1.7%	3,2%	1.4%	0.9%	1.9%	0.8%	0.5%	1.5%
40_30_3	5.5%	2.9%	8.6%	3,2%	2,2%	4.4%	2.8%	1.8%	3.9%
40_60_1	5.4%	3.8%	7,2%	4.1%	2.6%	5.5%	1.9%	1.3%	2.7%
40_60_3	14.5%	11.0%	17.8%	11.6%	9.5%	13.6%	7.4%	6,2%	8.6%
40_90_1	4.6%	3,2%	6.3%	3.1%	2.2%	4.2%	2.9%	2.2%	4.0%
40_90_3	13.4%	10.2%	16.6%	10.0%	7.9%	12.2%	9.1%	7.5%	10.3%
All	**4.8%**	**3.5%**	**6.3%**	**3.6%**	**2.7%**	**4.6%**	**2.5%**	**1.9%**	**3.2%**

the optimal solutions if known, or to the best found upper bounds otherwise. These computational results demonstrate the efficiency of the proposed method.

Futures works will include the development of efficient exact approaches. One can think about a column generation process embedded in a branch-and-price to solve formulation $[P2]$ more efficiently. Constraints (31)-(33) must then be generated on the fly. However, a simultaneous column and row generation is a complex process and it is not clear that the results will be significantly improved. One can also observe an intrinsic decomposition of the problem into a scheduling problem on one hand and into a resource management problem on the other hand. This lead to investigate a branch-and-check approach as well as cut generation processes. Last but not least, we have only addressed the deterministic problem, but, as a matter of fact, the problem is stochastic by nature.

References

Baptiste, P., Le Pape, C., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*. Kluwer Academic Publishers.

Budai, G., Dekker, R., and Nicolai, R. (2008). Maintenance and Production: A Review of Planning Models. In *Complex System Maintenance Handbook*, Springer Series in Reliability Engineering, pages 321–324. Springer London.

Carlos, S., Sánchez, A., Martorell, S., and Marton, I. (2013). Onshore wind farms maintenance optimization using a stochastic model. *Mathematical and Computer Modelling*, 57(7-8):1884–1890.

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

De Reyck, B., Demeulemeester, E., and Herroelen, W. (1998). Local search methods for the discrete time/resource trade-off problem in project networks. *Naval Research Logistics (NRL)*, 55(6):553–578.

Ding, F., Tian, Z., and Jin, T. (2013). Maintenance modeling and optimization for wind turbine systems: A review. In *2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE)*, pages 569–575.

Froger, A., Gendreau, M., Mendoza, J., Pinson, E., and Rousseau, L.-M. (2016). Maintenance scheduling in the electricity industry: A literature review. *European Journal of Operational Research*, 251(3):695–706.

Giorsetto, P. and Utsurogi, K. (1983). Development of a New Procedure for Reliability Modeling of Wind Turbine Generators. *IEEE Transactions on Power Apparatus and Systems*, PAS-102(1):134–143.

Kovács, A., Erds, G., Viharos, Z., and Monostori, L. (2011). A system for the detailed scheduling of wind farm maintenance. *CIRP Annals - Manufacturing Technology*, 60(1):497–501.

Malapert, A., Guéret, C., and Rousseau, L.-M. (2012). A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research*, 221(3):533 – 545.

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

Pisinger, D. and Ropke, S. (2010). Large Neighborhood Search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US.

Prud'homme, C., Fages, J.-G., and Lorca, X. (2014). *Choco3 Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.

Rodriguez, J. (2007). A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B: Methodological*, 41(2):231–245.

Ropke, S. and Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *Principles and Practice of Constraint Programming-CP 1998*, pages 417–431.

Appendix A. Notations

Time

- $\mathcal{T}$: time horizon
- $\mathcal{D}$: set of days
- $\mathcal{T}_d$: set of periods belonging to day $d \in \mathcal{D}$
- t_d^{rest} : last period $t \in \mathcal{T}$ before the rest period following day $d \in \mathcal{D}$

Locations

- $\mathcal{L}$: set of locations (wind farms and technician home depots)
- $\delta_{ll'} = \begin{cases} 1 & \text{if a technician is allowed to work at both locations } l \text{ and } l' \text{ during the same day,} \\ 0 & \text{otherwise.} \end{cases}$

Technicians

- $\mathcal{S}$: set of skills
- $\mathcal{R}$: set of technicians
- $\lambda_{rs} = \begin{cases} 1 & \text{if technician } r \in \mathcal{R} \text{ masters skill } s \in \mathcal{S}, \\ 0 & \text{otherwise.} \end{cases}$
- $\pi_r^t = \begin{cases} 1 & \text{if technician } r \in \mathcal{R} \text{ is available during period } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$
- l_r^t : location of technician $r \in \mathcal{R}$ when he or she is not available at period $t \in \{t' \in \mathcal{T}, \pi_r^{t'} = 1\}$

Tasks

- $\mathcal{I}$: set of tasks to be performed on the turbines
- $ov(\mathcal{I})$: family of sets of tasks that cannot overlap
- l_i : location where task $i \in \mathcal{I}$ has to be performed
- $\mathcal{M}_i$: set of execution modes for task $i \in \mathcal{I}$
- m_i^0 : execution mode related to the outsourcing of task $i \in \mathcal{I}$ ($m_i^0 \in \mathcal{M}_i$)
- q_{im} : number of technicians required at each period to perform task $i \in \mathcal{I}$ in mode $m \in \mathcal{M}_i$ ($q_{im_i^0} = 0$)
- d_{im} : duration of task $i \in \mathcal{I}$ if performed in mode $m \in \mathcal{M}_i$ ($d_{im_i^0} = 0$)
- $\gamma_i^t = \begin{cases} 1 & \text{if task } i \in \mathcal{I} \text{ can be executed at } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$
- s_i : skill required to perform task $i \in \mathcal{I}$
- o_i : penalty if task $i \in \mathcal{I}$ is outsourced.

Turbines

- $\mathcal{W}$: set of turbines
- $b_{wi} = \begin{cases} 1 & \text{if the execution of task } i \in \mathcal{I} \text{ shuts down the turbine } w \in \mathcal{W} \\ & \text{when technicians are effectively working on } i, \\ 0 & \text{otherwise.} \end{cases}$
- $\widetilde{b}_{wi} = \begin{cases} 1 & \text{if the execution of task } i \in \mathcal{I} \text{ shuts down the turbine } w \in \mathcal{W} \\ & \text{during the rest periods it overlaps,} \\ 0 & \text{otherwise.} \end{cases}$
- g_w^t : profit if turbine $w \in \mathcal{W}$ can produce energy at period $t \in \mathcal{T}$
- $\widetilde{g}_w^d$: profit if turbine $w \in \mathcal{W}$ can produce energy during the rest period following day $d \in D$

Plans

- $\mathcal{P}$: set of plans
- $\mathcal{P}_i$: set of plans involving task $i \in \mathcal{I}$
- i_p : task involved in plan $p \in \mathcal{P}$
- $a_p^t = \begin{cases} 1 & \text{if task } i_p \text{ is executed in period } t \in \mathcal{T} \\ 0 & \text{otherwise.} \end{cases}$
- q_p : number of required technicians if plan $p \in \mathcal{P}$ is selected
- $\mathcal{R}_p = \mathcal{R}_{i_p}$
- $b_{wp} = b_{wi_p}$
- $\widetilde{b}_{wp} = \widetilde{b}_{wi_p}$
- $o_p = o_{i_p}$

Appendix B. Data generation

An instance of the problem is characterized by:

- a finite time horizon,
- a set of wind farms,
- a set of turbines distributed over the wind farms,
- a set of maintenance tasks to schedule,
- a set of technicians to perform the tasks,
- profits generated by the turbines that are available to produce energy,
- outsourcing penalties.

The generator is based on the following parameters:

- $|\mathcal{T}|$, $|\mathcal{D}|$, $|\mathcal{L}|$, $|\mathcal{I}|$, $|\mathcal{S}|$ (length of time horizon, number of days, number of wind farms, number of tasks, and number of skills),
- $Dn_{\mathcal{W}}^{\mathcal{L}}$: distribution of the number of turbines per wind farm,
- $Dn_{\mathcal{I}}^{\mathcal{W}}$: distribution of the number of tasks per turbine,
- $Dn_{\mathcal{S}}^{\mathcal{I}}$: distribution of the number of skills required by each task,
- p^{inc} : probability that a technician is not allowed to work at two different wind farms during the same day,

- p_{SBL}, p_{BL}, p_{SST}, p_{ST}, p_{NS} : probability for a task to shutdown respectively all the turbines of a wind farms even during the rest periods; only the turbine on which the task is executed even during the rest periods; all the turbines of a wind farms except during the rest periods; only the turbine on which the task is executed except during the rest periods; no turbines,
- Di_{dur} : distribution of the duration of a task if performed by a single technician,
- Di_{req} : distribution of the number of technicians that can perform a task at the same time,
- $Dr_{\#skills}$: distribution of the number of skills mastered by a technician,
- $Dr_{\#unv}$: distribution of the number of periods during which a technician is unavailable,
- Dw_{power} : distribution of the nominal power (in kW) of each turbine,
- $\bar{\omega}$: average wind-speed values on each wind farm,
- $\bar{\omega}_{max}$: maximal wind-speed value allowed to perform a task.

We generate an instance following multiple steps. We first fix the number of wind farms, turbines and tasks. We then build the matrix $\left(\delta_{l,l'}\right)_{(l,l')\in\mathcal{L}^2}$ taking into account the geographical distribution of the wind farms and the parameter p^{inc}.

For each task $i \in \mathcal{I}$, according to the distribution Di_{dur}, we generate a duration dur_i if only one technician performs the task. Then, we sort based on Di_{req} a minimal number n_i^{min} and a maximal number n_i^{max} of technicians that can perform i at any given period. Based on these two values, we build the set $\mathcal{M}_i$ of execution modes by assuming that the duration of a task is proportional to the number of technicians assigned to the task (i.e., the duration is set to $\lfloor \frac{dur_i}{n} + 0.5 \lfloor$ if n technicians are assigned to i). We create therefore $n_i^{max} - n_i^{min} + 1$ execution modes, as well as an additional mode corresponding to the outsourcing of i. Since we round values, some execution modes can have the same duration with different requirements. We then perform a postprocessing procedure such that the set $\mathcal{M}_i$ meets the two following requirements:

- $\forall m, m' \in \mathcal{M}_i$, $q_{im} \neq q_{im'}$,
- $\forall m, m' \in \mathcal{M}_i$, $q_{im} < q_{im'} \Leftrightarrow d_{im} > d_{im'}$.

We also randomly generate the skills required by task i according to $Dn_{\mathcal{S}}^{\mathcal{I}}$. Arbitrarily, we build $ov(\mathcal{I})$ considering that overlapping tasks are forbidden on the same turbine.

After generating the tasks, we compute the number of technicians that has to be associated with each skill. More specifically, for each skill we first sum all the duration of the tasks (that require this particular skill) if they are executed by only one technician. The result is then divided by the product of the length of the time horizon and the average number of skills mastered by a technician (based on Dr_{skills}). Being an empirical method, we have noted that to multiply this final number by a factor of $\epsilon = 1.2$ allows to schedule at least 80% of the tasks. A technician is first created with a single mastered skill. We add randomly more skills to each technician according to $Dr_{\#skills}$. Finally, we generate his or her availability schedule using $Dr_{\#unv}$.

As it concerns the profits generation, the main purpose is to use realistic values. Therefore, it is mainly based on the work of (Giorsetto and Utsurogi, 1983). First, we generate wind-speed values ω_l^t for every wind farm $l \in \mathcal{L}$ and every period $t \in \mathcal{T}$ using a Rayleigh distribution based on $\bar{\omega}$. This implies independence between wind forecasts from a period to another one. This assumption is not always verified in practice. To smooth out the speed-values, a 3-moving average is used. The resulting values compare well to real data. Afterwards, we compute the parameters $(\gamma_i^t)_{t\in\mathcal{T},i\in\mathcal{I}}$ according to the value of $\bar{\omega}_{max}$. Then, we set for each turbine its nominal power P_w based on Dw_{power}. The power P_w^t generated at each period t by every turbine $w \in \mathcal{W}$ is then computed using the formula from (Giorsetto and Utsurogi, 1983) :

$$P_w^t = \begin{cases} 0 & \text{if } \omega_{l_w}^t < VCI \\ P_w \left(A + B\omega_{l_w}^t + C\left(\omega_{l_w}^t\right)^2 \right) & \text{if } VCI \leq \omega_{l_w}^t < VR \\ P_w & \text{if } VR \leq \omega_{l_w}^t < VCO \\ 0 & \text{if } \omega_{l_w}^t > VCO \end{cases} .$$

where VCI, VR and VCO are respectively the cut-in, the rated, and the cut-out wind speed and l_w denotes the location of turbine w. We compute constants A,B, and C with the following equations:

$$\begin{aligned} A &= \frac{1}{(VCI - VR)^2} \left(VCI\,(VCI + VR) - 4VCI \cdot VR \left(\frac{VCI + VR}{2VR} \right)^3 \right) \\ B &= \frac{1}{(VCI - VR)^2} \left(4\,(VCI + VR) \left(\frac{VCI + VR}{2VR} \right)^3 - 3\,(3VCI + VR) \right) \\ C &= \frac{1}{(VCI - VR)^2} \left(2 - 4 \left(\frac{VCI + VR}{2VR} \right)^3 \right) \end{aligned} .$$

The last step consists in computing the profit values from the power P_w^t generated by the turbine $w \in \mathcal{W}$ at each period $t \in \mathcal{T}$. Let $\hat{g}$ be the profit generated by the production of 1 kWh and h^{day} an estimation of the number of hours per period. We compute the profit g_w^t generated by each turbine $w \in \mathcal{W}$ that is available at period $t \in \mathcal{T}$ as follows:

$$g_w^t = \hat{g} \cdot h^{day} \cdot P_w^t. \tag{B.1}$$

The same process is repeated for the rest periods. Finally, we compute outsourcing penalties multiplying the maximal duration of a task by the the maximal power that could be generated by a turbine in one period. With this definition, outsourcing a task is almost always non-profitable.

Table B.7 presents the detail parameter setting used in the generation process.

Table B.7: Detail parameter setting used by the instance generator

Parameters	**Values**					
$(\lvert\mathcal{T}\rvert, \lvert\mathcal{D}\rvert)$	$(20, 10)$			$(40, 10)$		
$(\lvert L\rvert, \lvert I\rvert)$	(6, 30)	(10, 60)	(15, 90)	(6, 30)	(10, 60)	(15, 90)
$\mathcal{D}i_{dur}$	$\mathcal{U}(4, 10)$			$\mathcal{U}(4, 20)$		
$\lvert S\rvert$	1 or 3					
$Dn_{\mathcal{W}}^{\mathcal{L}}$	$\mathcal{U}(1, 6)$					
$Dn_{\mathcal{I}}^{\mathcal{W}}$	$\mathcal{U}(1, 3)$					
$Dn_{\mathcal{S}}^{\mathcal{I}}$	1					
p^{inc}	0.3					
$(p_{SBL}, p_{BL}, p_{SST}, p_{ST}, p_{NS})$	$(0.05, 0.25, 0.05, 0.25, 0.15)$					
Di_{req}	$\mathcal{U}(1, 4)$					
$Dr_{\#skills}$	$\mathcal{U}(1, 3)$					
$Dr_{\#unv}$	$\mathcal{U}(1, 4)$					
Dw_{power}	$\mathcal{U}(1500, 2500)$					
$\bar{\omega}$	5					
$\bar{\omega_a}$	12					
(VCI, VR, VCO)	$(3, 14, 25)$					