Les Cahiers du GERAD ISSN: 0711–2440

Real-time bi-objective personnel re-scheduling in the retail industry

R. Hassani,
G. Desaulniers, I. El Hallaoui

G–2019–55

July 2019

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

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

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2019
– Library and Archives Canada, 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Real-time bi-objective personnel re-scheduling in the retail industry

Rachid Hassani [a,b]

Guy Desaulniers [a,b]

Issmail El Hallaoui [a,b]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

rachid.hassani@gerad.ca
guy.desaulniers@gerad.ca
issmail.elhallaoui@gerad.ca

July 2019
Les Cahiers du GERAD
G–2019–55

Abstract: Personnel scheduling consists of determining least-cost work schedules to cover the demand of multiple jobs expressed in number of employees per job and period of a given horizon. During the operations, minor disruptions to the planned schedule, such as employee lateness, often occur and must be addressed in real time without changing too much the schedule. In this paper, we develop a fast re-scheduling heuristic that can be used to correct such minor disruptions in a retail industry context where employees can be assigned to a wide variety of shifts, starting and ending at numerous times. This heuristic can compute a set of approximate Pareto-optimal solutions that achieve a good compromise between cost and number of shift changes. It can be seen as a labeling algorithm that partially explores a network defined by the edges of the convex hull of the solutions of an integer program. Theoretical insights are provided to support certain speedup rules. Computational experiments conducted on instances derived from real-life datasets involving up to 95 employees show the heuristic efficiency. In less than one second on average, it can compute Pareto-optimal solutions for more than 98% of the tested scenarios.

Keywords: Personnel scheduling, real-time re-scheduling, bi-objective, labeling heuristic

Acknowledgments: This work was funded by Kronos Canadian Systems, Prompt, and the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant #RDC 530544–18. This financial support is greatly appreciated. The authors are also grateful to the personnel of Kronos for describing the problem and providing datasets.

1 Introduction

Personnel scheduling arises in many areas such as retail, health, postal services, transportation and industrial production. The personnel scheduling process can vary from one area to another but the goal, generally, remains the same. It consists of constructing the work schedule of a set of employees over a given time horizon to satisfy the demand in employees induced by a set of tasks or jobs. The computed schedule must respect various working, union and legislative constraints, and must be optimal with respect to one or several selection criteria such as the employee salaries, the quality of the work provided or the employee preferences. Finding such a schedule is a very delicate exercise which is better performed using an optimization software.

In practice, personnel scheduling is subject to a great deal of uncertainty. Indeed, employees may be late, absent or the observed demand may also differ from the expected demand for certain periods of the horizon. These incidents, called minor disruptions, are revealed dynamically during the operations. When such a disruption occurs, the planned schedule becomes infeasible and must be updated, generally in real time, to retrieve a feasible schedule. Usually, for a minor disruption, the re-optimized schedule deviates very little from the planned schedule, i.e., it is obtained by applying only a few changes to the current schedule. This is very desirable in practice: the planned work shifts cannot be changed for a large number of employees just because one of them is late. The quality of the new schedule should, therefore, be evaluated with respect to two criteria: the number of modifications made to the planned schedule and the cost generated by these modifications. This leads to a real-time bi-objective personnel re-scheduling problem (RBPRP).

In this paper, we aim at developing an effective heuristic for solving the RBPRP where the demand in employees can vary frequently during the day and the number of possible work shifts is very large. In particular, we have in mind problems arising in the retail industry where the set of feasible shifts may include shifts that can start at various times of the day (e.g., at every 15 minutes between 7h00am and 6h00pm) and last various durations (e.g., between 3 and 9 hours, by increment of 15 minutes). Furthermore, given its variability, the demand is not specified by shifts but rather by short time periods (e.g., 15-minute periods). The proposed heuristic must be fast and provide to the store manager a small set of updated schedules that are Pareto-optimal with respect to the number of changes brought to the planned schedule and the costs. From this set, the manager can then choose the schedule which seems the best from his/her point of view. The cost of a new schedule is computed by taking into account the management and operating costs on the day of the disruption as well as future costs to be incurred from additional schedule changes that may be needed on the subsequent days to rebalance the employee schedules and avoid weekly overtime as much as possible.

1.1 Literature review

Since the 1950s, personnel scheduling has been widely studied in the operations research literature as shown in the comprehensive surveys [16, 3, 4]. Research on personnel re-scheduling is much more recent as it began in the early 2000s. Most works have addressed the nurse re-scheduling problem which considers a very limited number of possible shifts (for example, from 7am to 3pm, from 3pm to 11pm, and from 11pm to 7am). Furthermore, the demand for a given task is often expressed as a number of nurses required by shift. In this context, a minor disruption corresponds to the absence of one or several nurses for a whole shift. Moz and Pato [12, 13] proposed exact branch-and-bound algorithms to solve this problem. The computational times of these algorithms being disproportionately long, they are not applicable in real time. Various heuristic algorithms dealing with the same problem have also been developed: see the works of Moz and Pato [12, 14], Pato and Moz [15], Maenhout and Vanhoucke [11], and Kitada et al. [9, 10]. Although some of these heuristics provide fast computational times, they are more relevant to the health field than to the retail domain which offers much more flexibility on the possible shifts and on the allowable changes to adjust the planned schedule. Note that larger disruptions in a health care setting (induced by a revision of the demand and offer of a whole 8-hour shift) have also been treated by exact branch-and-bound and branch-and-price algorithms in Bard and Purnomo [1, 2].

In 2016, Gross et al. [6] worked on re-scheduling doctors in a hospital, after the absence of one of them, by seeking a compromise between the quality of the new schedule and the distance of the latter from the initial schedule. In their setting, the number of work shifts (or duties) is equal to 17, demand is expressed by shift and the planning horizon spans 28 days. They formulated their problem as a mixed integer linear program that can be solved using a commercial branch-and-cut algorithm. They report computational time of up to 21 seconds. Note that to achieve the best compromise between the two objectives, the program was run several times with different parameter choices.

In 2015, Froger [5] studied a personnel re-scheduling problem arising in a retail context identical to our study except that large disruptions (e.g., when a relatively large variation of the demand is expected over one or several days due to bad weather forecasts or an unplanned sale) are considered instead of minor ones. She modeled the problem as a mixed integer linear program where the variables are defined for each planned shift of each employee and each possible transformation of these shifts according to some modification rules. This work has many similarities with ours. Indeed, the initial personnel scheduling problem is the same and the changes that can be made to the scheduled shifts are also the same. However, the size of the disruptions differs and the time available to deal with them is much less important in our case (a few seconds against a few minutes). Therefore, the solution approach developed by Froger [5] cannot be applied to a minor disruption in real time.

To the best of our knowledge, no works have been published to handle a minor disruption in a retail store or in a similar context, with the exception of the recent paper by Hassani et al. [7]. They introduced a heuristic that relies on the dual information obtained while solving the initial planning problem. This information is updated using a regression method after each time that the schedule is adjusted following a disruption. The proposed heuristic returns a small set of re-scheduling options that impact only the day of the disruption, ensuring that the rest of the schedule remains feasible but not necessarily optimal or near-optimal.

1.2 Contributions

In this paper, we introduce a new heuristic for the RBPRP that is characterized by efficiency and speed. This heuristic re-optimizes the planned schedule in real time when a minor disruption occurs with the bi-objective of minimizing costs and the number of required changes. Contrarily to what was proposed in Hassani et al. [7], the changes here can affect the day of the disruption as well as the subsequent days. Using basic decisions that can generate all possible schedules, our heuristic aims at finding decision sequences, called policies, that lead to Pareto-optimal solutions. All feasible policies can be modeled using a network where each vertex represents a feasible solution and each arc a basic decision. Based on theoretical insights, we significantly reduce the part of this network that is explored to yield a fast heuristic. The resulting heuristic is tested on RBPRP instances derived from real-life datasets and involving up to 95 employees. In more than 98% of the test cases, it can find all the Pareto-optimal solutions in an average computational time of less than one second.

1.3 Paper structure

This paper is organized as follows. In Section 2, we define the RBPRP and formulate it as a constrained shortest path problem. In Section 3, we first present some theoretical results that support the design of the proposed heuristic and then describe this heuristic. In Section 4, we report and analyze the computational results obtained on RBPRP instances derived from real-world datasets. Finally, conclusions are drawn in Section 5.

2 Problem definition and formulation

In this section, we begin by stating the RBPRP. Then, we introduce some terminology and concepts that are necessary to formulate the problem. Finally, we formulate it as a constrained shortest path problem and highlight the relationship between the underlying network and the convex hull of the set of feasible solutions.

2.1 Definition of the RBPRP

To define the RBPRP, we start by stating the personnel scheduling problem that needs to be solved to establish an initial planned schedule. Let $\mathcal{H}$ be a planning horizon (typically, one week or one month), where the days are numbered from 1 to $|\mathcal{H}|$. $\mathcal{H}$ is discretized in periods of equal length (say, 15 minutes) which form the set $\mathcal{P}$, numbered from 1 to $|\mathcal{P}|$. Furthermore, let $\mathcal{W}$ be a set of jobs that need to be covered in each period of $\mathcal{P}$ with a given number of employees that may depend on the period. To cover these demands, a set of skilled employees $\mathcal{E}$ is available. Each employee $e \in \mathcal{E}$ is qualified to work only on a subset of the jobs $\mathcal{W}_e$. For each employee $e \in \mathcal{E}$, a subset of possible mono-job shifts $\mathcal{S}_e$ on which he/she can work is computed a priori by a heuristic procedure and given as an input. A shift $s \in \mathcal{S}_e$ is defined by a starting period $b_s \in \mathcal{P}$, an end period $f_s \in \mathcal{P}$, and a job $w_s \in \mathcal{W}_e$. We denote by $l_s(= f_s - b_s + 1)$ its length, h_s its assignment day (i.e., that containing period b_s), and $e_s \in \mathcal{E}$ its associated employee. Furthermore, for $e \in \mathcal{E}$ and $h \in \mathcal{H}$, we define $\mathcal{S}_e^h \subseteq \mathcal{S}_e$ as the (possibly empty) subset of shifts that can be assigned to employee e on day h.

The planning problem consists of finding feasible work schedules for the employees in $\mathcal{E}$ such that they cover at least cost the demands of each job in $\mathcal{W}$ over the horizon $\mathcal{H}$. A feasible schedule for an employee e is composed of a subset of the available shifts in $\mathcal{S}_e$ such that a certain number of working rules are satisfied. For example, the schedule of each employee must contain at most one shift per day, a minimum number of days off per week, a minimum rest time between two consecutive work shifts and a maximum number of worked hours. Under-coverage of a demand at a given period (i.e., assigning fewer employees than requested) is prohibited but can be avoided by scheduling highly penalized anonymous shifts to be dispatched later to temporary employees. Demand over-coverage is accepted but also penalized. The cost of a complete schedule is computed as the sum of these penalties and labor costs. These costs do not correspond to the employee salaries (otherwise, minimizing costs would result in less working hours for the more experienced employees) but ensure through non-decreasing stepwise functions that the total working time is more or less balanced among the employees. A detailed definition of this problem over a one-week horizon is given in Hassani et al. [7].

The RBPRP must be solved when a minor disruption occurs and makes the planned schedule infeasible. Such a disruption can result, for example, from the lateness or the absence of an employee, the increase/decrease in the demand of a job for a limited number of periods, or an unforeseen event that prevents an employee to finish a work shift. In this paper, we focus on a minor disruption triggered by the lateness of an employee as most of the other situations can be treated similarly (see Hassani et al. [7]). A disruption due to a lateness is characterized by:

- the employee $\hat{e} \in \mathcal{E}$ who is late;
- the day $\hat{h} \in \mathcal{H}$ when the disruption occurs;
- the planned shift $\hat{s} \in \mathcal{S}_{\hat{e}}^{\hat{h}}$ of employee $\hat{e}$ on day $\hat{h}$;
- the lateness duration $\hat{l}$ in periods ($\hat{l} < l_{\hat{s}}$);
- the period $\hat{p} \in \mathcal{P}$ at which the manager is notified that employee $\hat{e}$ will be late ($\hat{p} \leq b_{\hat{s}}$).

This disruption is denoted $\hat{\Delta} = (\hat{e}, \hat{h}, \hat{s}, \hat{l}, \hat{p})$. When it occurs, the planned schedule becomes infeasible because shift $\hat{s}$ cannot be operated as planned and re-scheduling must be performed to retrieve an updated feasible solution. Thus, the RBPRP consists of finding a new feasible solution to the original planning problem such that all operated shifts up to period $\hat{p}$ are fixed and all other planned shifts can be modified including those on days $\hat{h}+1, \hat{h}+2, \ldots, |\mathcal{H}|$. Note that a planned shift that is ongoing at period $\hat{p}$ but not finished can also be changed after period $\hat{p}$. Hence, the new schedule must be feasible with respect to the constraints of the original planning problem. In addition, two objectives should be sought. First, the new schedule should be similar to the planned schedule in terms of the number of modified shifts. Second, the additional cost incurred by the modifications should be minimized. Given that these two objectives might be contradictory, the RBPRP falls into the class of bi-objective optimization problems and a small set of Pareto-optimal solutions should be produced. Finally, because delay between period $\hat{p}$ and the disruption starting period $b_{\hat{s}}$ may be very small, even

null, re-scheduling must often be performed in real time (i.e., some Pareto-optimal solutions must be found in less than a few seconds).

2.2 Concepts and terminology

The initial personnel scheduling problem can be modeled as the integer linear program (ILP) proposed by Hassani et al. [7] and presented in details in Appendix A. For ease of exposition and because every anonymous shift selected during planning is subsequently assigned to a temporary employee for the operations, we discard the anonymous shifts from our discussion. Let $x = ((x_e^h)_{\substack{h\in\mathcal{H}\\ e\in\mathcal{E}}}, x^r)^\top$ be the vector of variables, where x_e^h represents a sub-vector of binary variables of dimension $|\mathcal{S}_e^h|$ such that $x_e^h(s) = 1$ if the proposed shift $s \in \mathcal{S}_e^h$ is assigned to employee e on day h and 0 otherwise. Furthermore, x^r is a sub-vector containing all the remaining integer variables that are necessary to compute the penalties and the labor costs. The variables in x^r are totally dependent on the binary shift variables $x^b = (x_e^h)_{\substack{h\in\mathcal{H}\\ e\in\mathcal{E}}}$ when the sum of the costs and penalties is minimized. Let $\mathcal{X}$ be the set of feasible solutions to ILP and denote by $c(x) = c^\top x$ its objective function. Consequently, ILP writes as: $\min_{x\in\mathcal{X}} c^\top x$.

Because any feasible solution $x \in \mathcal{X}$ describes a feasible work schedule for all employees in $\mathcal{E}$, x is also called a schedule afterwards. For a schedule $x \in \mathcal{X}$, denote by $s_x(e, h)$ the shift assigned to employee $e \in \mathcal{E}$ on day $h \in \mathcal{H}$ in this schedule. By breach of terminology, we say that an employee is assigned on day $h \in \mathcal{H}$ to a nil shift denoted $s_0(h)$ if h is a day off for this employee. In this case, we have $s_x(e, h) = s_0(h)$. For the sake of notational conciseness, we assume that vector x_e^h has an additional component for this nil shift.

Consider an optimal planned schedule $x_0 \in \mathcal{X}$. When a disruption $\hat{\Delta} = (\hat{e}, \hat{h}, \hat{s}, \hat{l}, \hat{p})$ occurs and the corresponding RBPRP must be solved, at least another solution $x \in \mathcal{X}$ such that $x_{\hat{e}}^{\hat{h}}(\hat{s}) = 0$ (among other restrictions) must be found. Observe that both solutions x_0 and x belong to the same domain $\mathcal{X}$. To find a solution x from a starting solution x_0, we consider the following three types of decisions g^S, g^X and g^O that we call the *generating decisions*:

- $g^S(s_1, s_2)$: Swap the employees for shifts s_1 and s_2;
- $g^X(s_1, s_2, q)$: Extend shift s_2 to cover the first, or the last, q periods of shift s_1 which is shortened by the same number of periods;
- $g^O(s, e, h)$: If $s \neq s_0(h)$, assign shift s on day h to employee e who was assigned to a day off; otherwise ($s = s_0(h)$), assign a day off on day h to employee e and deleting his/her planned shift on day h.

Note that any such decision may assign an infeasible shift s to an employee e (i.e., $s \notin \mathcal{S}_e$). To reach a feasible schedule, this shift needs to be replaced in a subsequent decision.

In the following, we use a vector $(g_1, g_2, \ldots, g_m)$ to denote a sequence of m generating decisions that are applied in the order g_1 to g_m. This order is important because the decisions are not commutative. For example, consider three shifts s_1, s_2 and s_3 that are assigned to three employees e_1, e_2 and e_3, respectively, on the same day and which can be swapped by them. Swapping first the shifts of e_1 and e_2 before swapping the shifts of e_2 and e_3 results in a solution where e_1 is assigned to s_2, e_2 to s_3, and e_3 to s_1. On the other hand, swapping first the shifts of e_2 and e_3 before swapping those of e_1 and e_2 yields a different solution where e_1 is assigned to s_3, e_2 to s_1 and e_3 to s_2. Because some shifts are replaced by others in each generating decision, we denote by $r(s)$ the shift that is assigned to employee e_s on day h_s after applying at least one decision involving a shift s. With this notation, the sequence of decisions in the previous two examples would be written as $(g^S(s_1, s_2), g^S(r(s_2), s_3))$ and $(g^S(s_2, s_3), g^S(s_1, r(s_3)))$, respectively.

Applying a sequence of m generating decisions from the initial schedule x_0 can yield a sequence of schedules $x_0, x_1, x_2, \ldots, x_m$ that might not be all feasible. Nevertheless, the following observation highlights the usefulness of the generating decisions.

Observation 1 *Let $x, y \in \mathcal{X}$. Schedule y can always be obtained from schedule x by applying a finite sequence of generating decisions.*

Indeed, for each employee $e \in \mathcal{E}$ and each day $h \in \mathcal{H}$ such that $s_x(e,h) \neq s_y(e,h)$, one can add to the sequence the decisions $g^O(s_0(h), e, h)$, if $s_x(e,h) \neq s_0(h)$, and $g^O(s_y(e,h), e, h)$, if $s_y(e,h) \neq s_0(h)$, which first assigns a day off to employee e on day h and then reassigns employee e to shift $s_y(e,h)$. Note that the resulting sequence proposed is trivial but other sequences using the other two decision types g^S and g^X might exist to move from x to y. In fact, in the proposed heuristic, the decision type g^O is used with parsimony.

Algebraically, the application of a sequence of generating decisions (possibly a single one) can be seen as a transition vector d between a solution x and another solution $y = x + d$, i.e., $d = y - x$. As mentioned above, the obtained solution y might be infeasible. Consequently, we are not interested in all sequences of generating decisions, but only in those that yield a feasible solution. The following definition introduces minimal ones.

Definition 1 *Given a solution $x \in \mathcal{X}$, a sequence of m generating decisions $(g_1, g_2, \ldots, g_m)$, represented by its transition vector d, is called an* elementary decision (from x) *if $x + d \in \mathcal{X}$ and the sequence is minimal in the sense that any of its prefixes $(g_1, g_2, \ldots, g_n)$ with $n < m$ leads to an infeasible solution.*

An elementary decision can also be represented by a transition vector $d = (d^b, d^r)^\top$ with $d^b = (d_e^h)_{\substack{h \in \mathcal{H} \\ e \in \mathcal{E}}}$. If this elementary decision does not change the shift assigned to an employee e on a day h, then all components of the sub-vector d_e^h are equal to 0. Otherwise, this sub-vector contains two non-zero components: the component associated with the initial shift is equal to -1, whereas that associated with the new shift is equal to 1. Note that a transition vector carries less information than the corresponding elementary decision as it does not keep track of the sequence of the generating decisions. In fact, several elementary decisions may yield the same transition vector.

Let $\mathcal{D}(x)$ be the set of elementary decisions from solution x. Each decision $d \in \mathcal{D}(x)$ generates a number of modifications n_d and a cost c_d which are given by:

$$n_d = |Supp^+(d^b)| \quad \text{and} \quad c_d = c^\top d,$$

where $Supp^+(u) = \{j \in J \mid u_j > 0\}$ denotes the index subset of the positive-valued components in a vector $u = (u_j)_{j \in J}$. Consequently, n_d counts one modification each time that the final assignment of an employee on a day does not correspond to its initial assignment. It is independent on the sequence of generating decisions in the sense that, if the sequence modifies the assignment several times, it is still counted as a single modification. The cost c_d is equal to the cost difference between the solutions x and $x + d$. It can be positive, negative or null.

To illustrate these concepts, let us consider the following example that involves four employees assigned to the shifts s_1, s_2, s_3 and $\hat{s}$ on a given day $\hat{h}$ (see Figure 1). We assume that these shifts cover the same job and that the period length is 15 minutes although the figure indicates the time in hours. Furthermore, the working rules defining the feasibility of a shift may be specific to each employee and specify that employee $e_{\hat{s}}$ cannot work more than 8 hours per day and employee e_{s_2} must work exactly 8 hours per day. No specific restrictions are considered for employees e_{s_1} and e_{s_3}.

The following disruption occurs on day $\hat{h}$ and becomes known at period $\hat{p}$ at around 9h45am: the employee $e_{\hat{s}}$ who was scheduled to start the shift $\hat{s}$ at 12pm will be late by 8 periods, i.e., he/she will arrive at 2pm as shown by the red block in Figure 1. The following four elementary decisions $d_1, d_2, d_3, d_4 \in \mathcal{D}(x_0)$ can be applied to handle this disruption. The changes made by these decisions are illustrated in green in Figures 2 to 5.

- $d_1 = g^X(\hat{s}, s_1, 8)$ with $n_{d_1} = 2$: It consists of extending shift s_1 to cover the 8 periods of lateness of employee $e_{\hat{s}}$ (see Figure 2).

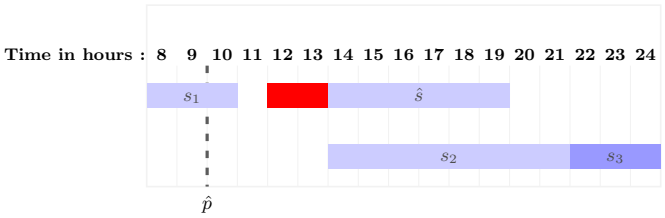

Figure 1: Example of a schedule x_0 disrupted on day $\hat{h}$

- $d_2 = (g^X(\hat{s}, s_2, 8), g^X(r(s_2), s_3, 8))$ with $n_{d_2} = 3$: It consists to start shift s_2 8 periods earlier to cover the lateness of employee $e_{\hat{s}}$ and then to cover the last 8 periods of the modified shift $r(s_2)$ by starting shift s_3 earlier (see Figure 3). Note that applying only $g^X(\hat{s}, s_2, 8)$ does not produce a feasible solution because employee e_{s_2} must work exactly 8 hours.
- $d_3 = g^S(\hat{s}, s_2)$ with $n_{d_3} = 2$: It consists of swapping the initial shift assignments of employees $e_{\hat{s}}$ and e_{s_2} (see Figure 4).
- $d_4 = (g^X(\hat{s}, s_1, 4), g^X(r(\hat{s}), s_2, 4), g^X(r(s_2), s_3, 4)$ with $n_{d_4} = 4$: It consists of extending shift s_1 to cover the first 4 periods of lateness of employee $e_{\hat{s}}$ before moving forward the start of shift s_2 for 4 periods in order to cover the 4 remaining periods of lateness. Finally, to assign only 8 hours of work to e_{s_2}, the last 4 periods of his/her modified shift $r(s_2)$ are removed from this shift and covered by starting shift s_3 4 periods earlier (see Figure 5).

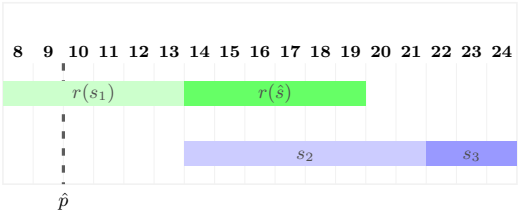

Figure 2: Elementary decision d_1

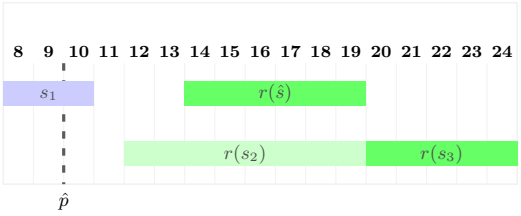

Figure 3: Elementary decision d_2

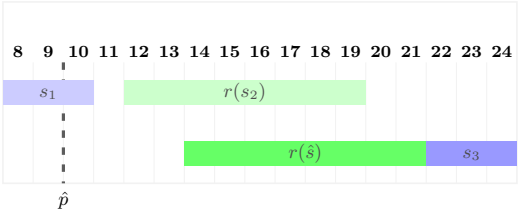

Figure 4: Elementary decision d_3

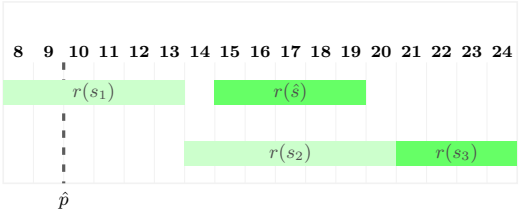

Figure 5: Elementary decision d_4

When a disruption occurs during the operations, an elementary decision similar to those presented above and that often impacts only the day of the disruption must be made as quickly as possible to rectify the disruption. If the planned schedule is optimal, the cost of this decision is often strictly positive because it typically yields additional working time for certain employees (unbalancing the working time between the employees) or additional demand over-coverage. To reduce this cost increase, a sequence of elementary decisions (possibly involving shift modifications on multiple days) can be applied. Such a decision sequence must generate a sequence of feasible solutions and aims at achieving a tradeoff between cost and total number of modifications. These decision sequences are called admissible policies.

Definition 2 *Given a schedule $x \in \mathcal{X}$, a sequence of m elementary decisions $(d_1, d_2, \ldots, d_m)$ is called an* admissible policy *for x if*

$$d_i \in \mathcal{D}(x + \sum_{k=1}^{i-1} d_k), \qquad \forall i \in \{1, \ldots, m\}.$$

The set of all admissible policies for a schedule x is denoted $\Pi(x)$.

According to Observation 1, we deduce that, when a disruption $\hat{\Delta} = (\hat{e}, \hat{h}, \hat{s}, \hat{l}, \hat{p})$ occurs and perturbs an initial schedule x_0, any feasible solution in $\mathcal{X} \setminus \{x_0\}$ can be reached using a policy in $\Pi(x_0)$. However, the manager is not interested by all these solutions but rather by a subset of so-called admissible solutions, denoted $\hat{\mathcal{X}}$. A solution in $\hat{\mathcal{X}}$: i) corrects the disruption $\hat{\Delta}$; ii) does not contain any over-coverage that can be avoided by simply reducing the length of a shift or removing a complete shift; and iii) respects the planned schedule x_0 up until period $\hat{p}$. The set of all admissible policies for x_0 that conduct to the solutions in $\hat{\mathcal{X}}$ is denoted $\hat{\Pi}(x_0)$. To take these restrictions into account, we also limit the set of elementary decisions to those yielding an admissible solution in $\hat{\mathcal{X}}$.

To evaluate an admissible policy $\delta = (d_1, d_2, \ldots, d_m) \in \hat{\Pi}(x_0)$ composed of m elementary decisions, we introduce the two criteria:

$$\varphi^c(\delta) = \sum_{k=1}^{m} c_{d_k} \qquad \text{and} \qquad \varphi^n(\delta) = |Supp^+(\sum_{k=1}^{m} d_k)|,$$

which compute the total cost and the total number of modifications incurred by policy δ, respectively. Note that $\varphi^n(\delta)$ is not necessarily equal to $\sum_{k=1}^{m} n_{d_k}$ because the shift or day off of an employee on a given day can be modified by more than one elementary decision in δ.

Criteria φ^c and φ^n are inherently conflicting. Indeed, the more we admit modifications, the more we are likely to lower the cost. We introduce the following strict ordering relation between two policies which is denoted by the symbol $\prec$:

$$\delta_1 \prec \delta_2 \iff \big(\varphi^c(\delta_1) \leq \varphi^c(\delta_2)\big) \wedge \big(\varphi^n(\delta_1) \leq \varphi^n(\delta_2)\big) \wedge \big((\varphi^c(\delta_1), \varphi^n(\delta_1)) \neq (\varphi^c(\delta_2), \varphi^n(\delta_2))\big).$$

Relation $\prec$ is clearly a partial ordering relation, in which case, two policies may not be comparable. This leads us to introduce the concepts of dominated and Pareto-optimal policies.

Definition 3 *Given a schedule $x \in \mathcal{X}$, a policy $\delta \in \hat{\Pi}(x)$ is said to be* dominated *if there exists a policy $\delta' \in \hat{\Pi}(x)$ such that $\delta' \prec \delta$. When there exist no such policies, policy δ is said to be* Pareto-optimal.

The set of Pareto-optimal admissible policies for a schedule x is denoted $\hat{\Pi}^*(x)$. Applying a dominated (resp. Pareto-optimal) policy in $\hat{\Pi}^*(x_0)$ to the planned schedule $x_0 \in \mathcal{X}$ produces a dominated (resp. Pareto-optimal) solution. The set of Pareto-optimal solutions forms the so-called Pareto front, which is denoted $\hat{\mathcal{X}}^*$ and can contain a relatively large number of solutions.

In practice, any solution of $\hat{\mathcal{X}}^*$ with a relatively high cost or number of modifications will never be adopted by the manager. For this reason, the manager can set a search zone parameterized by two non-negative parameters Φ^c and Φ^n that provide upper bounds on the criteria φ^c and φ^n, respectively. Consequently, for a given planned scheduled $x_0 \in \mathcal{X}$, the RBPRP is equivalent to searching in the set of admissible policies $\hat{\Pi}(x_0, \Phi^c, \Phi^n) = \{\delta \in \hat{\Pi}(x_0) \mid \varphi^c(\delta) \leq \Phi^c \wedge \varphi^n(\delta) \leq \Phi^n\}$, the set of Pareto-optimal policies, denoted $\hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$, that can be applied to produce Pareto-optimal solutions respecting the conditions fixed by the manager. The set of these solutions is denoted $\hat{\mathcal{X}}^*(x_0, \Phi^c, \Phi^n)$.

2.3 Problem formulation

The RBPRP consists of finding sequences of elementary decisions forming Pareto-optimal admissible policies in $\hat{\Pi}(x_0, \Phi^c, \Phi^n)$ that describes the Pareto front. It can be formulated on a network $\mathcal{G} = (\mathcal{V}, \mathcal{A})$,

where $\mathcal{V}$ is its vertex set and $\mathcal{A}$ its arc set. Set $\mathcal{V}$ contains a source vertex $o = v(x_0)$ representing the initial solution x_0, a sink vertex t, and one vertex $v(x)$ for each solution $x \in \hat{\mathcal{X}}$. In $\mathcal{A}$, there is an arc a between two vertices $v(x_i)$ and $v(x_j)$ if and only if there exists an elementary decision d_a from solution x_i such that $x_i + d_a = x_j$. This arc $a = (v(x_i), v(x_j))$ is associated with a cost $\gamma_a = c_{d_a}$. Furthermore, there is an arc a with $\gamma_a = 0$ between every vertex $v(x)$, $x \in \hat{\mathcal{X}}$, and the sink vertex t. Note that there are no arcs entering the source vertex o because x_0 is infeasible and, therefore, cannot be reached using an elementary decision.

An $o - t$ path $\rho = (o, v(x_1), v(x_2), \ldots, v(x_m), t)$ in network $\mathcal{G}$, also denoted by its arc sequence $\langle a_1, a_2, \ldots, a_{m+1} \rangle$, represents an admissible policy $\delta_\rho = (d_1, d_2, \ldots, d_m)$ in $\hat{\Pi}(x_0)$. Its cost is given by $\varphi^c(\delta_\rho) = \sum_{k=1}^{m+1} \gamma_{a_k}$ and its number of modifications by $\varphi^n(\delta_\rho) = |Supp^+(\sum_{k=1}^{m} d_k)|$. Policy δ_ρ belongs to $\hat{\Pi}(x_0, \Phi^c, \Phi^n)$ if $\varphi^c(\delta_\rho) \leq \Phi^c$ and $\varphi^n(\delta_\rho) \leq \Phi^n$. Furthermore, it belongs to the set of Pareto-optimal admissible policies $\hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$ if there exists no other $o-t$ path ρ' representing an admissible policy $\delta_{\rho'}$ such that $\varphi^c(\delta_{\rho'}) \leq \varphi^c(\delta_\rho)$, $\varphi^n(\delta_{\rho'}) \leq \varphi^n(\delta_\rho)$ and $(\varphi^c(\delta_{\rho'}), \varphi^n(\delta_{\rho'})) \neq (\varphi^c(\delta_\rho), \varphi^n(\delta_\rho))$. Thus, the RBPRP consists of finding Pareto-optimal $o-t$ paths ρ in $\mathcal{G}$ such that $\varphi^c(\delta_\rho) \leq \Phi^c$ and $\varphi^n(\delta_\rho) \leq \Phi^n$. This may be seen as equivalent to solving a shortest path problem with resource constraints (SPPRC) on $\mathcal{G}$ by dynamic programming (see, e.g., Irnich and Desaulniers [8]), where the objective is to minimize path cost under the constraint that the number of modifications should not exceed Φ^n. However, in our case, all paths reaching a vertex $v(x)$ are equivalent because both criterion φ^c and φ^n only depend on the solution x and not the path used to reach it.

Network $\mathcal{G}$ is not acyclic. However, even if there are negative arc costs, there is no need to impose elementarity requirements in this SPPRC because every cycle in $\mathcal{G}$ incurs no cost and no modification and can, thus, be discarded. The main difficulty with this SPPRC formulation is, thus, the size of network $\mathcal{G}$. In practice, it cannot be built a priori as it requires the enumeration of all admissible solutions in $\hat{\mathcal{X}}$. In Section 3, we introduce a heuristic that explores only a part of this network which is built dynamically.

2.4 Relationship with the set of admissible solutions

To conclude Section 2, we highlight a relationship between network $\mathcal{G}$ and the convex hull of the set of admissible solutions $\hat{\mathcal{X}}$. Recall that a solution $x \in \hat{\mathcal{X}}$ is written as $x = (x^b, x^r)^\top$, where $x^b = (x_e^h)_{\substack{h \in \mathcal{H} \\ e \in \mathcal{E}}}$ is a vector of binary shift variables and x^r is dependent on x^b. Therefore, $\hat{\mathcal{X}}$ is bounded and its convex hull $conv(\hat{\mathcal{X}})$ is a polytope.

Proposition 1 *$x \in \hat{\mathcal{X}}$ if and only if x is an extreme point of $conv(\hat{\mathcal{X}})$.*

Proof. See Appendix B □

From this proposition, we deduce that, in network $\mathcal{G}$, every vertex in $\mathcal{V} \setminus \{o, t\}$ corresponds to an extreme point of the polytope $conv(\hat{\mathcal{X}})$. The following proposition indicates that there is a bijection between the edges of this polytope and the arcs linking two vertices in $\mathcal{V} \setminus \{o, t\}$.

Proposition 2 *Let $x, y \in \hat{\mathcal{X}}$ be two extreme points of the polytope $conv(\hat{\mathcal{X}})$. These extreme points are adjacent in $conv(\hat{\mathcal{X}})$ if and only if there exists an elementary decision $d \in \mathcal{D}(x)$ such that $y = x + d$.*

Proof. See Appendix C □

Propositions 1 and 2 show that network $\mathcal{G}$ enables the exploration of $conv(\hat{\mathcal{X}})$ by moving along the edges of this polytope. The arcs exiting vertex o allow to move from the infeasible solution x_0 to admissible solutions in $conv(\hat{\mathcal{X}})$. Similarly as above, it can be proven that these admissible solutions are adjacent to x_0 in the polytope $conv(\mathcal{X})$ because they are obtained using elementary decisions from x_0. Finally, all arcs entering vertex t are used to gather admissible policies yielding different admissible solutions at a common vertex in order to find Pareto-optimal ones.

3 Solution algorithm

As mentioned in Section 2.3, solving the RBPRP is equivalent to solving a SPPRC defined on $\mathcal{G}$. However, $\mathcal{G}$ is so large in practice that it cannot be built a priori. Consequently, to solve the RBPRP, we propose a heuristic labeling algorithm that explores a relatively small subnetwork of $\mathcal{G}$ that is built dynamically. This heuristic is based on theoretical insights that are presented first.

3.1 Theoretical insights

Some results in this section (but not the proposed heuristic) are limited to a subset of the admissible policies in $\hat{\Pi}(x_0, \Phi^c, \Phi^n)$. This subset is denoted $\underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$ and composed of policies $\delta = (d_1, d_2, \ldots, d_m)$ such that each elementary decision d_k, $k = 1, \ldots, m$, is formed only of generating decisions of type g^O and the number of modifications strictly increases from one elementary decision to the next, i.e., $\varphi^n(\delta_i) < \varphi^n(\delta_{i+1})$ for all $i = 1, \ldots, m-1$, where $\delta_i = (d_1, d_2, \ldots, d_i)$ is the policy made up of the first i decisions of δ. As highlighted by the next proposition, focusing on this subset of policies is not restrictive.

Proposition 3 *Let $x_0 \in \mathcal{X}$ and $x \in \hat{\mathcal{X}}^*(x_0, \Phi^c, \Phi^n)$. Solution x can always be obtained from solution x_0 by applying a policy in $\underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$.*

Proof. See Appendix D. □

Denote by $\underline{\hat{\Pi}}^*(x_0, \Phi^c, \Phi^n) = \underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n) \cap \hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$ the subset of policies in $\underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$ which yield a Pareto-optimal solution in $\hat{\mathcal{X}}^*(x_0, \Phi^c, \Phi^n)$. Observe that, if there exist two policies $\underline{\delta} \in \underline{\hat{\Pi}}^*(x_0, \Phi^c, \Phi^n)$ and $\delta \in \hat{\Pi}^*(x_0, \Phi^c, \Phi^n) \setminus \underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$ that allows to move from solution x_0 to the same Pareto-optimal solution x, then $\varphi^c(\underline{\delta}) = \varphi^c(\delta)$ and $\varphi^n(\underline{\delta}) = \varphi^n(\delta)$. Consequently, for any policy in $\hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$, there exists one in $\underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$ that yields the same solution. Thus, to find Pareto-optimal solutions, it is sufficient to only look for policies in $\underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$.

Let $\delta_c^*, \delta_n^* \in \underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)$ be two policies such that

$$\delta_i^* \in \underset{\delta \in \underline{\hat{\Pi}}(x_0, \Phi^c, \Phi^n)}{\arg\min} \varphi^i(\delta), \qquad i \in \{c, n\}.$$

Thus, δ_c^* and δ_n^* are policies that minimize the cost and the number of modifications, respectively. By definition, they both belong to $\underline{\hat{\Pi}}^*(x_0, \Phi^c, \Phi^n)$ and can be used to restrict the search space as shown in Figure 6. In this figure, the dots represent Pareto-optimal policies. The dashed rectangle shows the initial search space where such policies can be found. The rectangles in gray and black represent the regions containing the policies dominated by δ_c^* and δ_n^*, respectively. Finding these policies may, thus, reduce significantly the search space. Finally, note that policies δ_c^* and δ_n^* are not necessarily unique. However, in the following, we use δ_c^* and δ_n^* to denote representative policies.

The following proposition shows that policy δ_n^* can easily be found.

Proposition 4 *There exists an elementary decision $d \in \mathcal{D}(x_0)$ such that $\delta_n^* = (d)$.*

Proof. See Appendix E □

Finding policy δ_c^* is much more difficult. However, if we find a policy δ such that $\varphi^c(\delta) = 0$, then $\delta = \delta_c^*$ because x_0 is optimal for ILP and, thus, $\varphi^c(\delta_c^*) \geq 0$. In this case, Φ^n can be replaced by $\varphi^n(\delta_c^*)$ to speed up the algorithm.

For any policy $\delta = (d_1, d_2, \ldots, d_m) \in \underline{\hat{\Pi}}^*(x_0, \Phi^c, \Phi^n)$ (even in $\hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$) with $m \geq 2$, there exists an index $l(\delta) \in \{2, 3, \ldots, m\}$ such that $c_{d_{l(\delta)-1}} \geq 0$ and $c_{d_k} < 0, \forall k \geq l(\delta)$. Otherwise, there would be a contradiction with the fact that δ is Pareto-optimal as removing the last decision would

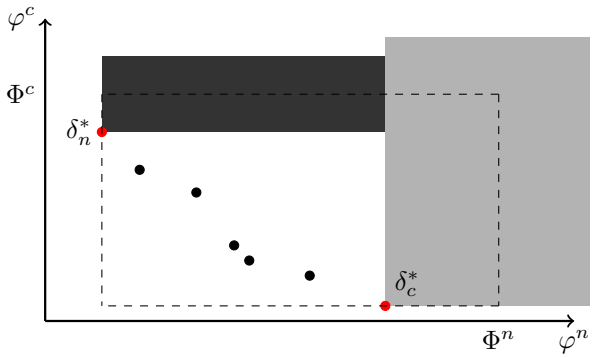

Figure 6: Restricted search space induced by policies δ_c^* and δ_n^*

yield a dominating policy. On the other hand, it is well known from linear programming theory that, from any non-optimal extreme point x_1 of a polyhedron, there exists at least one sequence of adjacent extreme points $(x_1, x_2, \ldots, x_m)$ of decreasing costs that reaches an optimal extreme point x_m. In our case, this sequence can be generated using elementary decisions $d_2, d_3, \ldots, d_m$ with costs $c_{d_k} < 0$, $k \in \{2, 3, \ldots, m\}$. A first elementary decision d_1 with a non-negative cost is also required to move from the infeasible solution x_0 to feasible solution x_1. Nevertheless, it is not clear if there exists such a policy $\delta = (d_1, d_2, \ldots, d_m)$ in the restricted set $\underline{\hat{\Pi}}^*(x_0, \Phi^c, \Phi^n)$. Therefore, based on these observations, we conjecture that, to find a set of Pareto-optimal solutions in $\hat{\mathcal{X}}^*(x_0, \Phi^c, \Phi^n)$ that describes the Pareto front, it is sufficient to only consider the policies $(d_1, d_2, \ldots, d_m) \in \hat{\Pi}^*(x_0, \Phi^c, \Phi^n)$ such that $c_{d_k} < 0$, $\forall k \geq 2$. The proposed heuristic restricts its search to such policies.

The existence of an elementary decision in a Pareto-optimal policy can be used to predict the existence of other decisions in the same policy. Indeed, if a decision forces an employee e_1 to work two hours more and an employee e_2 to work two hours less, the balance between these teo employees might be restored in a subsequent decision by making e_1 (resp., e_2) work less (resp., more). Such a decision is said to be *compatible* with the first decision made. Let us introduce a measure of compatibility.

For an elementary decision d (resp., policy δ), denote by $\mathcal{E}_d$ (resp., $\mathcal{E}_\delta$) the set of employees affected by this decision (resp., policy). Let $\delta = (d_1, d_2, \ldots, d_m) \in \hat{\Pi}(x_0, \Phi^c, \Phi^n)$ be an admissible policy that produces the feasible solution $x_\delta \in \hat{\mathcal{X}}(x_0, \Phi^c, \Phi^n)$. The following function $\mu(x_0, \delta, d)$ that returns a value in $[0, 1]$ provides a "measure" of the compatibility of each elementary decision $d \in \mathcal{D}(x_\delta)$ with respect to policy δ:

$$\mu(x_0, \delta, d) = \sqrt[m]{\frac{|\mathcal{E}_\delta \cap \mathcal{E}_d|}{|\mathcal{E}_\delta \cup \mathcal{E}_d|}} \cdot \frac{\min\limits_{d' \in \mathcal{D}(x_\delta)} \varphi^n(\delta \oplus d')}{\varphi^n(\delta \oplus d)} \cdot \prod_{e \in \mathcal{E}_\delta \cap \mathcal{E}_d} \frac{\min\{N_{x_0}(e), N_{x_\delta + d}(e)\}}{\max\{N_{x_0}(e), N_{x_\delta + d}(e)\}}, \tag{1}$$

where the symbol $\oplus$ indicates the concatenation of a decision to a policy, and $N_y(e)$ is equal to the number of periods worked by an employee $e \in \mathcal{E}$ in a schedule $y \in \hat{\mathcal{X}}$. For a decision $d \in \mathcal{D}(x_\delta)$, a value $\mu(x_0, \delta, d)$ close to 1 indicates that d is highly compatible with δ; at the opposite, a value $\mu(x_0, \delta, d)$ close to 0 means that d is highly incompatible with δ. Indeed, the first term is equal to 1 if the employees affected by decision d are all involved in policy δ, to 0 if none of them are, and to a value between 0 and 1 otherwise. The second term evaluates the impact of decision d on the criterion φ^n compared to the minimal impact over all possible decisions. A decision yielding a relatively large number of modifications compared to the others is disfavored. Finally, the third term measures the relative change in the total working time of each employee in $\mathcal{E}_\delta \cap \mathcal{E}_d$. The focus is put on these employees instead of all employees in $\mathcal{E}_d$ because the largest changes mostly occur for the employees in the first decisions of a policy. A large relative change (either much more working time or much less) for an employee decreases the value of $\mu(x_0, \delta, d)$. Note that the mth root function applied in the first term is used to weight the first term against the others. Indeed, the more decisions are made, the more employees are likely to be involved in the modifications.

To extend a policy δ in the proposed heuristic, we consider only elementary decisions d that yield a sufficiently large value of function $\mu(x_0, \delta, d)$. In fact, in the following proposition, we show under an additional assumption often holding in practice that extending δ with a decision d such that $\mu(x_0, \delta, d) = 1$ is the best possible choice. This assumption stipulates that every elementary decision of a policy does not modify demand coverage for each job and each period. Indeed, in practice, the manager tries to cover all demands as planned so as to avoid a loss in service quality or a wage increase. We denote by $\hat{\underline{\Pi}}^{=}(x_0, \Phi^c, \Phi^n)$ the subset of policies in $\hat{\underline{\Pi}}(x_0, \Phi^c, \Phi^n)$ that respect this assumption.

Proposition 5 *Let $\delta = (d_1, d_2, \ldots, d_m) \in \hat{\underline{\Pi}}^{=}(x_0, \Phi^c, \Phi^n)$ be an admissible policy from x_0 that generates a solution x_δ. If there exists a decision $d \in \mathcal{D}(x_\delta)$ such that $\mu(x_0, \delta, d) = 1$ and $(\delta \oplus d) \in \hat{\underline{\Pi}}^{=}(x_0, \Phi^c, \Phi^n)$, then policy $\delta \oplus d$ is not dominated by any policy $\delta' = \delta \oplus (d_{m+1}, \ldots, d_{m+q}) \in \hat{\underline{\Pi}}^{=}(x_0, \Phi^c, \Phi^n)$ with $d_{m+1} \neq d$ and $q \geq 1$.*

Proof. See Appendix F □

3.2 Heuristic

In this section, we describe the heuristic that we propose to solve the RBPRP. This heuristic is a labeling algorithm that explores the network $\mathcal{G}$ defined in Section 2.3 but only partially. Indeed, to speed up the search, it eliminates admissible elementary decisions based on various rules. These rules are:

1. An elementary decision involves shifts of a same day.
2. An elementary decision contains at most n^g generating decisions ($n^g = 2$ for our tests).
3. Generating decisions of type g^O are only used to initially correct the lateness of employee $\hat{e}$ if no other decision types can.
4. A policy $\delta = (d_1, d_2, \ldots, d_m)$ is such that $t_{d_k} \leq t_{d_{k+1}}$ for all $k = 1, 2, \ldots, m-1$, where $t_d \in \mathcal{P}$ is the earliest starting period b_s of a shift s involved in decision d.
5. A policy δ yielding solution x_δ cannot be extended with an elementary decision $d \in \mathcal{D}(x_\delta)$ if $\mu(x_0, \delta, d) \geq \frac{\varphi^n(\delta \oplus d)}{\Phi^n} - \epsilon$, where ϵ is a small positive tolerance.
6. A policy δ containing at least one elementary decision and yielding a solution x_δ can only be extended with an elementary decision $d \in \mathcal{D}(x_\delta)$ such that $c_d < 0$.

Rules 5 and 6 are derived from the discussions made in Section 3.1. The heuristic may also reduce the value of Φ^n and the total number of policies explored when it finds a policy δ_c^* such that $\varphi^c(\delta_c^*) = 0$.

In the proposed re-scheduling heuristic, we use the concept of a label ℓ to store a policy $\delta(\ell)$ and its corresponding solution $x_{\delta(\ell)}$. The generated labels are grouped in sets $\mathcal{L}_k$, $k = 1, 2, \ldots, K$, that contain all labels associated with a policy with k elementary decisions, where K is a sufficiently large integer. The labels representing non-dominated solutions are kept in set Ω^*. By breach of terminology, a label is said to be dominated (resp. non-dominated) when its associated solution is.

We start by describing the heuristic main procedure, before presenting some of the embedded functions. The pseudo-code of this main procedure is given in Algorithm 1. First, in Step 2, it finds through function `findCorrectiveGeneratingDecisions` a set G of generating decisions that correct disruption $\hat{X}$. Because these decisions do not necessarily correspond to elementary decisions, function `findAdjacentSolutions` is called in Step 4 for each decision $d \in D$ to find elementary decisions starting with decision g. These elementary decisions form one-decision policies represented by labels that are stored in $\mathcal{L}_1$ in Step 6. The labels in set $\mathcal{L}_1$ are then transferred to set Ω^*. Next, in the while loop (Steps 9–13), the algorithm extends all generated labels in set $\mathcal{L}_i$, not only the non-dominated ones, because labels are typically associated with different solutions to which different elementary decisions can be further applied. In each iteration i, it first propagates (Step 11) each label $\ell \in \mathcal{L}_i$ using function `propagate` to possibly create multiple labels stored in set $\mathcal{L}_{i+1}$ and representing policies

Algorithm 1: Re-scheduling Heuristic

input : initial schedule x_0, minor disruption $\hat{X}$;
output: set of labels Ω^*;

1 $\mathcal{L}_k \leftarrow \emptyset, k = 1, 2, \ldots, K$;
2 $G \leftarrow$ `findCorrectiveGeneratingDecisions`$(x_0, \hat{X})$;
3 **foreach** $g \in G$ **do**
4 $\quad D \leftarrow$ `findAdjacentSolutions`$(x_0, \emptyset, g)$;
5 $\quad$ **foreach** $d \in D$ **do**
6 $\quad\quad \mathcal{L}_1 \leftarrow \mathcal{L}_1 \cup \{(x_d, d)\}$;
7 $\Omega^* \leftarrow \mathcal{L}_1$;
8 $i \leftarrow 1$;
9 **while** $\mathcal{L}_i \neq \emptyset$ **do**
10 $\quad$ **foreach** $\ell \in \mathcal{L}_i$ **do**
11 $\quad\quad \mathcal{L}_{i+1} \leftarrow \mathcal{L}_{i+1} \cup$ `propagate`(x_0, ℓ) ;
12 $\quad \Omega^* \leftarrow \Omega^* \cup \mathcal{L}_{i+1}$;
13 $\quad i \leftarrow i + 1$;
14 $\Omega^* \leftarrow$`applyDominance`(Ω^*) ;

that extends policy $\delta(\ell)$ with a single elementary decision in $\mathcal{D}(x_{\delta(\ell)})$. This loop stops when no more labels can be extended. Finally, a function `applyDominance` is to find the non-dominated labels in set Ω^* (Step 14).

Now, let us detail the three functions `findCorrectiveGeneratingDecisions`, `findAdjacentSolutions`, and `propagate`. The pseudo-code of the first function is given in Algorithm 2. This function produces a set of generating decisions G that correct disruption $\hat{X}$. For each employee $e \in \mathcal{E} \setminus \{\hat{e}\}$ qualified for job $w_{\hat{s}}$, it finds the generating decisions that modify its current assignment and can correct the lateness of employee $\hat{e}$. If e is assigned to a shift s on day $\hat{h}$ (Step 4), it verifies if shift s can be switched with shift $\hat{s}$ (Step 5) or extended to cover the lateness interval $[b_{\hat{s}}, b_{\hat{s}} + \hat{l}]$ (Step 7). In both cases, the decision is added to set G only if it is considered as a candidate, i.e., if the resulting shifts respect employee availability and qualifications, as well as the minimum rest time between consecutive shifts. Note that the duration of these shifts might not be valid, yielding an infeasible solution to be corrected with one or several subsequent generating decisions. On the other hand, if employee e is assigned to a day off and no decisions of type g^S or g^X have been generated yet (Step 9), then the algorithm looks for potential decisions of type g^O that would assign employee e to a shift of minimal duration covering the lateness interval. Such a decision is added to set G^O if it respects the conditions to be a candidate stated above as well as the minimum number of days off. Decisions in G^O are only added to set G in Step 14 if no decisions of type g^S or g^X have been found.

Algorithm 2: Function `findCorrectiveGeneratingDecisions`

input : initial schedule x_0; minor disruption $\hat{X}$;
output: set of generating decisions G ;

1 $G \leftarrow \emptyset$, $G^O \leftarrow \emptyset$;
2 **foreach** $e \in \mathcal{E} \setminus \{\hat{e}\}$ *such that* $w_{\hat{s}} \in \mathcal{W}_e$ **do**
3 $\quad s \leftarrow s_{x_0}(e, \hat{h})$;
4 $\quad$ **if** $s \neq s_0(\hat{h})$ **then**
5 $\quad\quad$ **if** $b_{\hat{s}} + \hat{l} \leq b_s$ *and* $g^S(s, \hat{s})$ *is candidate* **then**
6 $\quad\quad\quad G \leftarrow G \cup \{g^S(s, \hat{s})\}$;
7 $\quad\quad$ **if** $[b_{\hat{s}}, b_{\hat{s}} + \hat{l}] \cap [b_s, f_s] = \emptyset$, $w_s = w_{\hat{s}}$, $\hat{p} \leq f_s$ *and* $g^X(\hat{s}, s, \hat{l})$ *is candidate* **then**
8 $\quad\quad\quad G \leftarrow G \cup \{g^X(\hat{s}, s, \hat{l})\}$;
9 $\quad$ **else if** $G = \emptyset$ **then**
10 $\quad\quad$ **foreach** $s' \in \mathcal{S}_e^{\hat{h}}$ *of minimal duration such that* $[b_{\hat{s}}, b_{\hat{s}} + \hat{l}] \subseteq [b_{s'}, f_{s'}]$ **do**
11 $\quad\quad\quad$ **if** $g^O(s', e, \hat{h})$ *is candidate* **then**
12 $\quad\quad\quad\quad G^O \leftarrow G^O \cup \{g^O(s', e, \hat{h})\}$;
13 **if** $G = \emptyset$ **then**
14 $\quad G \leftarrow G^O$

Function `findAdjacentSolutions` is called in Step 4 of Algorithm 1 to create elementary decisions starting with a given generating decision. The pseudo-code of this function is presented in Algorithm 3. The elementary decisions are stored in set D. Set S contains sequences of generating decisions currently under investigation. When one of them does not yield a feasible solution, it is extended in various ways and the resulting sequences are added to the set S^+. This function starts with a single sequence, composed of the generating decision g_0, in set S. Then, it enters a repeat loop that checks for every sequence $\sigma \in S$ if the schedule of every employee is feasible (Step 5). This check is performed by function `employeesWithInfeasibleSchedule` which returns the set of employees E with an infeasible schedule. If this set is empty, the sequence σ is added to set D. Otherwise, function `lengthenSequence` is called in Step 9 to create new sequences obtained by appending an additional generating decision g to σ. Such a generating decision must correct the infeasibility of the schedule of at least one employee in E. The repeat loop stops when either all sequences do not need to be extended or the maximum number of generating decisions is reached.

Algorithm 3: Function `findAdjacentSolutions`

input : schedule x_δ, policy δ, generating decision g_0;
output: set of elementary decisions D;

1 $D \leftarrow \emptyset$, $k \leftarrow 1$, $S \leftarrow \{(g_0)\}$;
2 **repeat**
3 $S^+ \leftarrow \emptyset$;
4 **foreach** $\sigma \in S$ **do**
5 $E \leftarrow$ `employeesWithInfeasibleSchedule`(x_δ, σ) ;
6 **if** $E = \emptyset$ **then**
7 $D \leftarrow D \cup \{\sigma\}$;
8 **else**
9 $S^+ \leftarrow S^+ \cup$ `lengthenSequence`(x_δ, σ, E) ;
10 $S \leftarrow S^+$, $k \leftarrow k + 1$;
11 **until** $S = \emptyset$ *or* $k = n^g$;

Algorithm 4 describes function `propagate` which is called in Step 11 of Algorithm 1 to extend a label ℓ and create new ones that are stored in set $\mathcal{L}$. It starts by calling function `findCandidate ElementaryDecisions` (defined below) that produces a relatively large set D of candidate elementary decisions (Step 2). Then, each elementary decision $d \in D$ must pass three selection tests (Steps 4 and 5) to be retained for label creation. First, it must lead to a policy that does not exceed the maximum number of modified shifts. Second, the cost of decision d must be negative as stipulated in Rule 6. Finally, the μ function value for decision d must be sufficiently large according to Rule 5. When all these tests are met, a new label $(x_{\delta(\ell)\oplus d}, \delta(\ell) \oplus d)$ is created and added to set $\mathcal{L}$ (Step 6). Finally, upon finding a zero-cost policy $\delta(\ell) \oplus d$, the value of Φ^n might be updated in Step 8.

Algorithm 4: Function `propagate`

input : initial schedule x_0, label ℓ, small tolerance $\epsilon > 0$;
output: set of new labels $\mathcal{L}$;

1 $\mathcal{L} \leftarrow \emptyset$;
2 $D \leftarrow$ `findCandidateElementaryDecisions`(x_0, ℓ);
3 **foreach** $d \in D$ **do**
4 **if** $\varphi^n(\delta(\ell) \oplus d) \leq \Phi^n$ *and* $c_d < 0$ **then**
5 **if** $\mu(x_0, \delta(\ell), d) \geq \frac{\varphi^n(\delta(\ell)\oplus d)}{\Phi^n} - \epsilon$ **then**
6 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(x_{\delta(\ell)\oplus d}, \delta(\ell) \oplus d)\}$;
7 **if** $\varphi^c(\delta(\ell) \oplus d) = 0$ **then**
8 $\Phi^n \leftarrow \min\{\Phi^n, \varphi^n(\delta(\ell) \oplus d)\}$;

For a label ℓ and its solution x_{δ_ℓ}, function `findCandidateElementaryDecisions` builds a list of elementary decisions D that can potentially improve this solution by re-allocating as much as possible the same amount of working time to an employee as in its initial schedule. It loops over all employees affected by the current policy $\delta(\ell)$ and computes for each employee the difference ΔN in its working

Algorithm 5: Function `findCandidateElementaryDecisions`

input : initial schedule x_0, label ℓ;
output: set of elementary decisions D;

1 $D \leftarrow \emptyset$;
2 **foreach** $e \in \mathcal{E}_{\delta(\ell)}$ **do**
3 $\quad \Delta N \leftarrow N_{x_{\delta(\ell)}}(e) - N_{x_0}(e)$;
4 $\quad$ **if** $\Delta N \neq 0$ **then**
5 $\quad\quad G \leftarrow$ `findGeneratingDecisions`$(x_{\delta(\ell)}, e, \Delta N)$;
6 $\quad\quad$ **foreach** $g \in G$ **do**
7 $\quad\quad\quad D \leftarrow D \cup$ `findAdjacentSolutions`$(x_{\delta(\ell)}, g)$;

time between its initial schedule and its current schedule (Step 3). If this difference is not equal to zero, then all possible generating decisions of type g^S or g^X that can reduce (resp. increase) its working time if ΔN is positive (resp. negative) are enumerated through function `findGeneratingDecisions` in Step 5 and stored in set G. Then, for each decision $g \in G$, function `findAdjacentSolutions` is called to create elementary decisions starting with decision g.

Because it applies Rules 1 to 6, the proposed re-scheduling heuristic is fast and partially explores network $\mathcal{G}$ which is implicitly constructed simultaneously. It offers no guarantee that it will find a set of Pareto-optimal solutions that will describe the Pareto-front. However, as presented in the next section, our computational experiments showed that it is very efficient at this task.

4 Computational experiments

To assess the efficiency of the heuristic described in Algorithm 1 and denoted H in the following, we performed computational experiments on different datasets using sets of disruption scenarios. In these tests, we compared heuristic H with the following two solution algorithms:

- An exact algorithm E that is used as a reference on the quality of the solutions returned by H. It consists of solving by a commercial solver the initial planning model described in Appendix A, modified as follows. First, all shifts or parts of shifts scheduled prior to period $\hat{p}$ in the initial solution x_0 are fixed. Second, all variables associated with shifts of employee $\hat{e}$ that are in conflict with the late arrival are set to 0. Finally, a constraint is added to ensure that at most Φ^n shifts are changed from the initial solution x_0.
- A variant of H, referred to as heuristic R, that replaces the test on the μ function in Step 5 of the propagate function described in Algorithm 4 by a random test which allows the selection of exactly the same number of elementary decisions selected using the test on μ. This heuristic is used to confirm the usefulness of function μ.

All algorithms were implemented in C++. All tests were executed on a Linux machine equipped with an 8-core Intel Core i7 processor clocked at 3.4GHz and 16Gb of RAM. The mixed-integer programming solver Cplex, version 12.6.1.0, was used for algorithm E.

Below, we describe first the instances used for our experiments, including the procedure generating the disruption scenarios. Then, we present and analyze the computational results obtained.

4.1 Datasets and disruption scenarios

Our experiments were conducted using seven real-world datasets provided by our industrial partner. These datasets provide, for each period $p \in \mathcal{P}$ and job $w \in \mathcal{W}$, the number of employees required for job w in period p and, for each employee $e \in \mathcal{E}$ and day $h \in \mathcal{H}$, a set of potential shifts that can be assigned to employee e on day h. The characteristics of these datasets are given in Table 1. For each dataset, an optimal initial schedule was obtained by solving the integer linear program presented in Appendix A.

Table 1: Characteristics of the datasets

Dataset	No. employees	No. jobs	Horizon (in days)
I_1	15	5	7
I_2	25	5	7
I_3	29	6	7
I_4	32	3	7
I_5	47	6	7
I_6	49	7	7
I_7	95	7	7

For each dataset I_k, $k \in \{1, \ldots, 7\}$, we generated 30 scenarios, where each scenario is formed by a disruption $\hat{X} = (\hat{e}, \hat{h}, \hat{s}, \hat{l}, \hat{p})$ that perturbs the planned schedule. This disruption is generated as follows. First, day $\hat{h}$ is chosen using a uniform discrete distribution over horizon $\mathcal{H}$. Then, we choose employee $\hat{e}$ uniformly in the set of employees who work on day $\hat{h}$. The selection of $\hat{h}$ and $\hat{e}$ yields the shift $\hat{s}$. The lateness duration $\hat{l}$, in number of periods, is drawn from another discrete uniform distribution over the set $\{1, \ldots, \lfloor \frac{l_{\hat{s}}}{2} \rfloor\}$. Finally, the period $\hat{p}$, when the employer becomes aware of the disruption, is chosen from a discrete uniform distribution over the set $\{b_{\hat{s}} - 8, \ldots, b_{\hat{s}}\}$. Note that a check is made to ensure that a scenario is not repeated.

4.2 Computational results

To devise a fast heuristic H, several choices were made to limit the labeling process. Consequently, there is no guarantee that heuristic H can find a feasible solution even if one exists, especially if parameter Φ^n is set to a low value, which might be desirable to avoid changing the schedules of many employees just because one employee is late. We performed a first series of tests to determine how frequently heuristic H fails to find a feasible solution in function of Φ^n. For values of $\Phi^n \in \{2, 3, 4, 5, 6\}$, Table 2 reports the percentage of failures obtained for each dataset for the corresponding 30 disruption scenarios. This percentage is computed as $100\frac{(n^E - n^H)}{n^E}$, where n^E (resp. n^H) is the number of scenarios for which a feasible solution, with no more than Φ^n shift changes, was found by algorithm E (resp. H). These results show that heuristic H always succeeds to find feasible solutions except for a small number of scenarios when the number of shift changes is restricted to a maximum of $\Phi^n = 2$.

Table 2: Percentage of failures with heuristic H

Dataset	$\Phi^n = 2$	$\Phi^n = 3$	$\Phi^n = 4$	$\Phi^n = 5$	$\Phi^n = 6$
I_1	0	0	0	0	0
I_2	0	0	0	0	0
I_3	0	0	0	0	0
I_4	3.3	0	0	0	0
I_5	3.3	0	0	0	0
I_6	0	0	0	0	0
I_7	0	0	0	0	0
Average	0.9	0	0	0	0

Now, let us analyze the quality of the solutions produced by heuristic H on the scenarios for which it did not fail for any value of Φ^n, i.e., 30 scenarios for all datasets except I_4 to I_5, and 29 scenarios for these two datasets. For each algorithm $i \in \{E, H, R\}$, let $\delta_i^* \in \underset{\delta \in \mathcal{Z}(\Phi^n)}{\operatorname{argmax}} \varphi^n(\delta)$ be the least-cost policy it found in the zone $\mathcal{Z}(\Phi^n) = \{\delta \in \hat{\Pi}(x_0, \Phi^c, \Phi^n) | \varphi^n(\delta) \leq \Phi^n\}$ and denote by $\gamma^i(\Phi^n)$ the cost of this policy. Furthermore, let $\Gamma^i(\Phi^n) = \gamma^i(\Phi^n) - \gamma^E(\Phi^n)$ be the error made by heuristic $i \in \{H, R\}$ in the search area $\mathcal{Z}(\Phi^n)$. For a given dataset I_k, $k = 1, \ldots, 7$, the average (resp. standard deviation and maximum) of this error over the n_H scenarios for which heuristic H did not fail, is denoted $\Gamma^i_{avg}(I_k, \Phi^n)$ (resp. $\Gamma^i_{sd}(I_k, \Phi^n)$ and $\Gamma^i_{max}(I_k, \Phi^n)$) for heuristic $i \in \{H, R\}$. Finally, the overall error average, i.e., over all datasets, for a given heuristic i is denoted $\Gamma^i_{AVG}(\Phi^n)$.

The results achieved by heuristics R and H compared to the exact algorithm E are summarized in Table 3 for each value of $\Phi^n \in \{2,3,4,5,6\}$. These results reveal a high success rate for heuristic H, which succeeds to compute a least-cost solution for 96.6% to 99.5% of the scenarios for each tested value of Φ^n. Given that, with $\Phi^n = 6$, heuristic H also computes all the solutions generated with $\Phi^n = 2,3,4,5$, we deduce that it can generate Pareto-optimal solutions for almost all values of $\Phi^n \in \{2,3,4,5,6\}$. Furthermore, when it cannot find a Pareto-optimal solution for a given Φ^n value, the cost error is very small as highlighted by the Γ^H_{avg} values. Notably, observe that, when $\Phi^n \geq 3$, heuristic H finds a least-cost solution for most scenarios, in particular, for all scenarios except one when $\Phi^n = 6$. We believe that this is due to the fact that any solution $x \in \hat{\mathcal{X}}^*(x_0, \Phi^c, \Phi^n)$ can be reached in network $\mathcal{G}$ using several paths and that larger values of Φ^n yield more chances to find one of them. In contrast, the performance of heuristic R deteriorates as the value of Φ^n increases from 2 to 4 and remains stable for larger values. In fact, the first elementary decisions which are not affected by the random selection often allow to find least-cost solutions for $\Phi^n = 2,3$. Hence, the random selection only starts to play an important role for larger Φ^n values. The poor performance of heuristic R clearly shows the efficiency of function μ to reduce the number of elementary decisions explored.

Table 3: Percentage of optimal policies and errors for heuristics R and H

Dataset	$\Phi^n = 2$								$\Phi^n = 3$							
	heuristic R				heuristic H				heuristic R				heuristic H			
	$OPT(\%)$	Γ^R_{avg}	Γ^R_{sd}	Γ^R_{max}	$OPT(\%)$	Γ^H_{avg}	Γ^H_{sd}	Γ^H_{max}	$OPT(\%)$	Γ^R_{avg}	Γ^R_{sd}	Γ^R_{max}	$OPT(\%)$	Γ^H_{avg}	Γ^H_{sd}	Γ^H_{max}
I_1	100	0	0	0	100	0	0	0	100	0	0	0	100	0	0	0
I_2	100	0	0	0	100	0	0	0	100	0	0	0	100	0	0	0
I_3	100	0	0	0	100	0	0	0	100	0	0	0	100	0	0	0
I_4	83.3	0.31	0.72	1.88	83.3	0.31	0.72	1.88	80	0.46	1.01	3.96	100	0	0	0
I_5	93.1	0.27	1.15	5.95	96.5	0.07	0.37	1.98	93.1	0.27	1.15	5.95	96.5	0.07	0.37	1.98
I_6	100	0	0	0	100	0	0	0	10	5.36	1.81	5.95	100	0	0	0
I_7	76.6	6.87	13.43	36.74	96.6	0.33	1.81	9.95	40	9.23	13.47	13.47	96.6	0.04	0.24	1.33
Γ^i_{AVG}		1.21				0.1				2.19				0.02		
$OPT(\%)$	93.3				96.6				74.7				99.0			

Dataset	$\Phi^n = 4$								$\Phi^n = 5$							
	heuristic R				heuristic H				heuristic R				heuristic H			
	$OPT(\%)$	Γ^R_{avg}	Γ^R_{sd}	Γ^R_{max}	$OPT(\%)$	Γ^H_{avg}	Γ^H_{sd}	Γ^H_{max}	$OPT(\%)$	Γ^R_{avg}	Γ^R_{sd}	Γ^R_{max}	$OPT(\%)$	Γ^H_{avg}	Γ^H_{sd}	Γ^H_{max}
I_1	100	0	0	0	100	0	0	0	100	0	0	0	100	0	0	0
I_2	100	0	0	0	100	0	0	0	100	0	0	0	100	0	0	0
I_3	3.3	10.22	6.21	26.23	96.6	0.07	0.41	2.23	3.3	9.69	5.43	26.23	96.6	0.07	0.41	2.23
I_4	26.6	2.37	2.22	5.95	96.6	0.06	0.34	1.89	26.6	2.37	2.22	5.95	96.5	0.06	0.34	1.89
I_5	24.1	3.35	3.56	9.91	96.5	0.07	0.37	1.97	24.1	3.35	3.56	9.91	100	0	0	0
I_6	10.0	5.36	1.81	5.95	100	0	0	0	0	32.62	10.40	38.99	100	0	0	0
I_7	10.0	14.24	9.63	37.34	100	0	0	0	6.6	15.39	9.01	34.5	100	0	0	0
Γ^i_{AVG}		5.08				0.03				9.06				0.02		
$OPT(\%)$	39.1				98.5				37.2				99.0			

Dataset	$\Phi^n = 6$							
	heuristic R				heuristic H			
	$OPT(\%)$	Γ^R_{avg}	Γ^R_{sd}	Γ^R_{max}	$OPT(\%)$	Γ^H_{avg}	Γ^H_{sd}	Γ^H_{max}
I_1	100	0	0	0	100	0	0	0
I_2	100	0	0	0	100	0	0	0
I_3	3.3	9.69	5.43	26.23	96.6	0.07	0.41	2.23
I_4	23.3	2.51	2.54	7.90	100	0	0	0
I_5	24.1	3.35	3.56	9.91	100	0	0	0
I_6	0	34.01	7.90	38.99	100	0	0	0
I_7	3.3	16.28	39.07	9.18	100	0	0	0
Γ^i_{AVG}		9.41				0.01		
$OPT(\%)$	36.3				99.5			

Next, we investigate the impact of the value of parameter Φ^n on the quality of the least-cost solution obtained by heuristic H. To do so, for each dataset, scenario and value of $\Phi^n \in \{3,4,5,6\}$, we define $\mathcal{G}^H(2,\Phi^n) = 100\frac{(\gamma^H(2)-\gamma^H(\Phi^n))}{\gamma^H(2)}$ as the difference (gain) in percentage between the costs of the least-cost policies obtained by heuristic H when allowing Φ^n shift changes compared to only two shift changes. For a given dataset, the average gain over all scenarios is denoted $\mathcal{G}^H_{avg}(2,\Phi^n)$. Furthermore, for each dataset, we compute the number of scenarios $n^H(\Phi^n-1,\Phi^n)$, $\Phi^n = 3,4,5,6$, for which the cost of the solution obtained with at most Φ^n shift changes is better than that of the solution with at most Φ^n-1 changes. Table 4 reports the average gains and the number of scenarios with an improved cost obtained for all values of $\Phi^n \in \{3,4,5,6\}$ and all datasets I_3 to I_7. It turned out that, for the small datasets I_1 and I_2, no gain is realized for each scenario and each value of $\Phi^n \geq 3$, i.e., the least-cost policy always involves two shift changes. This is not the case for the other datasets where increasing the allowed number of shift changes up to six yields average gains varying between 33% and 100%, with an overall average gain of 75.4%. Note that an average gain of 100% (for datasets I_6 and I_7) means that a zero-cost (i.e., least-cost) solution has been found for all scenarios with a maximum of six shift changes. On the other hand, a gain of less than 100% does not mean that the algorithm does not perform well because least-cost solutions might have positive costs. From the results in Table 4, observe that, for datasets I_3, I_4 and I_5, most improvement occurred when switching from three to four allowed shift changes. For the other two datasets (I_6 and I_7), better solutions for several scenarios are found as the value of Φ^n increases until finding a least-cost solution for each scenario. All these results show that additional cost reduction can be achieved in many cases by modifying only a few additional shifts but that, in most cases, a very limited number of shift changes (up to six) is sufficient.

Table 4: Average gain $\mathcal{G}^H_{avg}(2,\Phi^n)$ in percentage

Dataset	$\mathcal{G}^H_{avg}(2,3)$	$\mathcal{G}^H_{avg}(2,4)$	$\mathcal{G}^H_{avg}(2,5)$	$\mathcal{G}^H_{avg}(2,6)$	$n^H(2,3)$	$n^H(3,4)$	$n^H(4,5)$	$n^H(5,6)$
I_3	0	73.4	73.4	73.4	0	30	0	0
I_4	7.3	67.5	67.5	70.7	7	23	1	1
I_5	0	32.5	33.1	33.1	0	22	1	0
I_6	35.6	35.6	96.9	100	30	0	30	7
I_7	30.2	82.1	90.8	100	19	18	11	17
Average	14.6	58.2	72.3	75.4	11.2	18.2	8.6	5

In Table 5, we provide, for each dataset, the average and maximum computational time (T_{avg} and T_{max}) over the 30 scenarios required by heuristic H when $\Phi^n = 6$ and by the exact algorithm E when $\Phi^n = 2$ to 6. For heuristic H, we observe an overall average time of 0.79 second and a maximum time of 4.13 seconds. These computational times are more than reasonable compared to the times required by the exact algorithm which takes on average 30 seconds for a given value of Φ^n, with a maximum exceeding two minutes in some cases. Note that, to generate a set of Pareto-optimal solutions, the exact algorithm needs to solve a modified ILP for each value of Φ^n resulting in a total average time of approximately 150 seconds.

Table 5: Average and maximum computational time for heuristic H and algorithm E

Dataset	Heuristic H		Algorithm E									
			$\Phi_n = 2$		$\Phi_n = 3$		$\Phi_n = 4$		$\Phi_n = 5$		$\Phi_n = 6$	
	$T_{avg}(s)$	$T_{max}(s)$	$T_{avg}(s)$	$T_{max}(s)$	$T_{avg}(s)$	$T_{max}(s)$	$T_{avg}(s)$	$T_{max}(s)$	$T_{avg}(s)$	$T_{max}(s)$	$T_{avg}(s)$	$T_{max}(s)$
I_1	0.03	0.04	12.28	14.04	12.81	14.59	12.16	15.23	12.95	35.84	12.95	19.53
I_2	0.02	0.05	8.27	12.73	9.32	26.36	8.77	10.47	8.55	11.32	8.73	19.29
I_3	1.39	4.13	53.99	59.86	69.73	133.17	70.82	104.15	74.87	117.22	73.3	143.01
I_4	0.42	1.33	10.49	14,85	12.79	18.63	13.74	18.38	14.54	20.85	14.69	20.75
I_5	0.46	1.72	26.33	30.96	31.76	41.38	34.66	44.11	35.95	52.63	35.4	45.5
I_6	1.78	2.51	36.16	38.06	36.57	44.82	36.98	46.46	38.34	51.19	37.98	48,65
I_5	1.44	3.67	29.82	32.84	36.23	47.86	39.13	54.17	40.06	65.96	41.56	70.1
Average	0.79		25.33		29.89		32.86		32.18		32.09	

Finally, to better understand the fast computational times obtained by heuristic H, we report in Table 6 the average numbers of policies that are treated before and after some tests in heuristic H when $\Phi^n = 6$: n_{avg}^{P0} is the average number of policies $\delta(\ell) \oplus d$ considered in the tests of Step 4 of the propagate function (see Algorithm 4), n_{avg}^{P1} is the average number of policies surviving these two tests, and n_{avg}^{P2} is the average number surviving the test in Step 5 and, thus, producing a label. Note that these statistics are provided only for the datasets that require more than two shift modifications to find a least-cost solution for some scenarios. From these results, we compute that, on average, 55% of the generated policies (n_{avg}^{P0}) are rejected by the tests in Step 4, whereas 65% of the remaining ones (n_{avg}^{P1}) do not survive the test in Step 5. These results show that the proposed tests are efficient at significantly reducing the number of policies to consider and, therefore, the computational times. As highlighted by the previous results, this computational effort reduction is not made at the expense of bad-quality solutions.

Table 6: Average numbers of policies treated before and after some tests in heuristic H

Dataset	n_{avg}^{P0}	n_{avg}^{P1}	n_{avg}^{P2}
I_3	1558.9	1039.6	58.6
I_4	219.1	89.4	62.5
I_5	572.6	158.8	48.6
I_6	3516.7	1700.7	393.8
I_7	722.2	246.5	63.2

5 Conclusion

In this paper, we considered the RBPRP in a retail context where employees can be assigned to a wide variety of shifts. To solve this problem, we developed a fast heuristic that aims at correcting a minor disruption by proposing a set of solutions that achieve a good compromise between the cost and the number of shift modifications. Computational experiments conducted on 210 disruption scenarios generated from real-world datasets showed the effectiveness of this heuristic: it can compute the exact Pareto-optimal solutions in less than one second on average for more than 98% of the test cases.

We believe that the framework of the proposed heuristic is generic and could be used to tackle other optimization problems. In fact, it allows the exploration of the extreme points of a polyhedron and, to adapt it to a different problem, would require to redefine the generating decisions and the μ function. Such an adaptation would constitute an interesting research avenue. Other opportunities include the design of a robust heuristic that would take into account, in a stochastic fashion, the potential future disruptions as well as the integration of this re-scheduling tool in a heuristic for solving the initial planning problem.

A Mathematical model of the personnel scheduling problem

In this section, we present the integer programming model proposed by Hassani et al. [7] for the initial personnel scheduling problem. This model assumes that the horizon has seven days, numbered from 1 to 7. We describe first the required notation that has not be defined in the main text.

$\mathcal{S}^A$: Set of anonymous shifts used to avoid under-coverage;
n^O: Minimum number of days off to assign to each employee;
n^R: Minimum number of periods of rest between two consecutive shifts assigned to the same employee;
d_w^p: Number of employees required for job w in period p;
a_{sw}^p: Binary parameter equal to 1 if shift s covers job w in period p and 0 otherwise;
$\mathcal{K}^L$: Set of steps in the step function defining the labor costs;
n_k^L: Number of periods on step $k \in \mathcal{K}^L$;
c_k^L: Unit penalty on step $k \in \mathcal{K}^L$ (with $c_k^L < c_{k+1}^L$);
$\mathcal{K}^V$: Set of steps in the step function defining the over-coverage penalties;
n_k^V: Number of periods on step $k \in \mathcal{K}^V$;
c_k^V: Unit penalty on step $k \in \mathcal{K}^V$ (with $c_k^V < c_{k+1}^V$);
c_s^Y: Penalty for each period in an anonymous shift s;
M: Large constant;
X_{es}^h: Binary variable equal to 1 if shift s is assigned to employee e on day h and 0 otherwise;
Y_s: Nonnegative integer variable equal to the number of times that anonymous shift s is used;
L_e^k: Nonnegative integer variable equal to the number of periods worked by employee e on step $k \in \mathcal{K}^L$;
V_w^{kp}: Nonnegative integer variable equal to the number of employees assigned to job w in over-coverage in period p on step $k \in \mathcal{K}^V$;
O_e^h: Binary variable equal to 1 if employee e has a day off on day h and 0 otherwise.

With this notation, the initial personnel scheduling problem can be modeled as the following integer program:

$$\text{Minimize} \quad \sum_{e\in\mathcal{E}}\sum_{k\in\mathcal{K}^L} c_k^L L_e^k + \sum_{s\in\mathcal{S}^A} c_s^Y l_s Y_s + \sum_{p\in\mathcal{P}}\sum_{w\in\mathcal{W}}\sum_{k\in\mathcal{K}^V} c_k^V V_w^{kp} \tag{2}$$

$$\text{subject to :} \quad \sum_{s\in\mathcal{S}_e^h} X_{es}^h + O_e^h = 1, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H} \tag{3}$$

$$\sum_{h\in H} O_e^h \geq n^O, \qquad \forall e\in\mathcal{E} \tag{4}$$

$$\sum_{h\in\mathcal{H}}\sum_{s\in\mathcal{S}_e^h} l_s X_{es}^h - \sum_{k\in\mathcal{K}^L} L_e^k = 0, \qquad \forall e\in\mathcal{E} \tag{5}$$

$$M\,O_e^{h+1} + \sum_{s\in\mathcal{S}_e^{h+1}} b_s X_{es}^{h+1} - \sum_{s\in\mathcal{S}_e^h} (f_s + 1 + n^R) X_{es}^h \geq 0, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H}\setminus\{7\} \tag{6}$$

$$\sum_{e\in\mathcal{E}}\sum_{h\in\mathcal{H}}\sum_{s\in\mathcal{S}_e^h} a_{sw}^p X_{es}^h + \sum_{s\in\mathcal{S}^A} a_{sw}^p Y_s - \sum_{k\in\mathcal{K}^V} V_w^{kp} = d_w^p, \qquad \forall p\in\mathcal{P}, w\in\mathcal{W} \tag{7}$$

$$X_{es}^h \in \{0,1\}, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H}, s\in\mathcal{S}_e^h \tag{8}$$

$$O_e^h \in \{0,1\}, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H} \tag{9}$$

$$L_e^k \in [0, n_k^L], \text{integer}, \qquad \forall e\in\mathcal{E}, k\in\mathcal{K}^L \tag{10}$$

$$Y_s \geq 0, \text{integer}, \qquad \forall s\in\mathcal{S}^A \tag{11}$$

$$V_w^{kp} \in [0, n_k^V], \text{integer}, \qquad \forall w\in\mathcal{W}, p\in\mathcal{P}, k\in\mathcal{K}^V. \tag{12}$$

Objective function (2) minimizes the sum of the labor costs and the penalties incurred by the anonymous shifts and the over-coverage. Constraints (3) ensure that, on each day, each employee is assigned to a shift or a day off. Constraints (4) impose a minimum of n^O days off for each employee. Constraints (5) distribute the periods worked by each employee on the various steps of $\mathcal{K}^L$. Constraints (6) enforce a rest of at least n^R periods between shifts assigned to the same employee on two consecutive days. Finally, constraints (7) guarantee that, for each job and each period, a sufficient number of employees or anonymous shifts covers the demand. They also allow the computation of the number of employees in over-coverage per step in $\mathcal{K}^V$.

B Proof of Proposition 1

Proof. ($\Leftarrow$) By the definition of a convex hull, all extreme points of any set Q belong to Q. ($\Rightarrow$) Assume that x is not an extreme point of $conv(\hat{\mathcal{X}})$. In this case, x can be written as a convex combination of two other solutions in $\hat{\mathcal{X}} \setminus \{x\}$, i.e., there exists $x_1, x_2 \in \hat{\mathcal{X}} \setminus \{x\}$ and $\alpha \in]0,1[$ such that $x = \alpha x_1 + (1-\alpha)x_2$. In particular, we have $x^b = \alpha x_1^b + (1-\alpha)x_2^b$. Because x^b, x_1^b and x_2^b are binary vectors, this equality implies that $x_1^b = x_2^b = x^b$. Since x_1^r, x_2^r and x^r depend on x_1^b, x_2^b and x^b, we get $x_1 = x_2 = x$, which contradicts $x_1, x_2 \in \hat{\mathcal{X}} \setminus \{x\}$. Consequently, the assumption is false and x is an extreme point of $conv(\hat{\mathcal{X}})$. □

C Proof of Proposition 2

Proof. ($\Rightarrow$) Assume that x and y are adjacent in $conv(\hat{\mathcal{X}})$. Therefore, they are the extreme points of an edge of $conv(\hat{\mathcal{X}})$. From the primal simplex algorithm, we know that, to move along the edge between x and y, a minimal set of shift variables indexed by S must be entered into the basis. This set defines a direction d such that $y = x + d$, where

$$d_j^b = \begin{cases} 1 & \text{if } j \in S \\ -1 & \text{if } j \in Supp^+(x^b) \setminus Supp^+(y^b) \\ 0 & \text{otherwise} \end{cases}$$

and d^r is derived from d^b. Each component $d_j^b = -1$ corresponds to a generating decision of type g^O that proposes a day off to an employee that was assigned to a shift on this day and each component $d_j^b = 1$ corresponds to a generating decision of type g^O that, at the opposite, assigns a shift to an employee that had a day off. Sequencing these decisions starting with the former ones yields an elementary decision because of the minimality of set S.

($\Leftarrow$) Assume that there exists an elementary decision $d = (d^b, d^r)^\top \in \mathcal{D}(x)$ such that $y = x + d$. Let $S = Supp^+(d^b)$. Because decision d is elementary, entering into the basis the shift variables indexed by S in the order provided by d allows to move directly from x to y and the move only occurs when the last of these variables is pivoted in. This shows that x and y are adjacent in the polytope $conv(\hat{\mathcal{X}})$. □

D Proof of Proposition 3

Proof. Let $d = x - x_0$ be the transition vector from x_0 to x. For each employee $e \in \mathcal{E}$ and day $h \in \mathcal{H}$ such that the sub-vector d_e^h is not null (starting with $\hat{e}$ and day $\hat{h}$), define the generating decisions $g^O(s_0(h), e, h)$ if $s_{x_0}(e,h) \neq s_0(h)$ (i.e., if $d_e^h(s_{x_0}(e,h)) = -1$) and $g^O(s_x(e,h), e, h)$ if $s_x(e,h) \neq s_0(h)$ (i.e., if $d_e^h(s_x(e,h)) = 1$) which allow to reassign employee e from shift $s_{x_0}(e,h)$ to shift $s_x(e,h)$ on day h and put them in a list in this order. Once all employees and days have been treated, this list forms a sequence of generating decisions of type g^O allowing a transition between x_0 and x. Scanning this list in order, one can identify a sequence of elementary decisions $d_1, d_2, \ldots, d_m$ by testing after each generating decision whether it yields a feasible solution or not. Note that $n_{d_i} \geq 1$ for all $i = 1, \ldots, m$. Indeed, for n_{d_i} to be equal to 0, all generating decisions in d_i would need to assign an employee to a day off on a given day. This is not possible for d_1 because this decision involves employee $\hat{e}$ who

receives a new shift on day $\hat{h}$. Moreover, this is not possible for the other decisions d_i, $i = 2, \ldots, m$, because it would mean that all employees assigned to a day off in this decision were in over-coverage during their whole shift and, therefore, the solution $x_0 + \sum_{k=1}^{i-1} d_k$ is not in $\hat{\mathcal{X}}$, i.e., decision d_{i-1} is not a feasible elementary decision.

Given that, for each employee e and day h, there is at most one vector d_i, $i = 1, \ldots, m$, with a positive component in the sub-vector d_e^h, we deduce that $\varphi^n(\delta_i) = \sum_{k=1}^{i} n_{d_k}$ for all $i = 1, \ldots, m$. Furthermore, because $n_{d_i} \geq 1$ for all $i = 1, \ldots, m$, we get that $\varphi^n(\delta_i) = \sum_{k=1}^{i} n_{d_k} < \sum_{k=1}^{i+1} n_{d_k} = \varphi^n(\delta_{i+1})$ for all $i = 1, \ldots, m-1$. Therefore, the sequence of elementary decisions $d_1, d_2, \ldots, d_m$ forms a policy in $\hat{\underline{\Pi}}(x_0, \Phi^c, \Phi^n)$ that can be applied to move from x_0 to x. □

E Proof of Proposition 4

Proof. Let $\delta = (d_1, d_2, \ldots, d_m) \in \hat{\underline{\Pi}}(x_0, \Phi^c, \Phi^n)$ with $m \geq 2$ and denote by $\delta' = (d_1, d_2, \ldots, d_{m-1}) \in \hat{\Pi}(x_0, \Phi^c, \Phi^n)$ the policy obtained by omitting the last decision d_m from δ. Because $\varphi^n(\delta') < \varphi^n(\delta)$ by the definition of set $\hat{\underline{\Pi}}(x_0, \Phi^c, \Phi^n)$, policy δ cannot have a minimum number of modifications, i.e., $\delta \neq \delta_n^*$. Consequently, δ_n^* must have a single elementary decision. □

F Proof of Proposition 5

Proof. Assume that there exists such an elementary decision d. Because $\mu(x_0, \delta, d) = 1$, we deduce that

$$\frac{\min\limits_{d' \in \mathcal{D}(x_\delta)} \varphi^n(\delta \oplus d')}{\varphi^n(\delta \oplus d)} = 1, \quad \sqrt[m]{\frac{|\mathcal{E}_\delta \cap \mathcal{E}_d|}{|\mathcal{E}_\delta \cup \mathcal{E}_d|}} = 1, \quad \frac{\min\{N_{x_0}(e), N_{x_\delta + d}(e)\}}{\max\{N_{x_0}(e), N_{x_\delta + d}(e)\}} = 1, \forall e \in \mathcal{E}_\delta \cap \mathcal{E}_d.$$

$$\Rightarrow \quad \begin{cases} \min\limits_{d' \in \mathcal{D}(x_\delta)} \varphi^n(\delta \oplus d') = \varphi^n(\delta \oplus d), \\ \mathcal{E}_\delta \cap \mathcal{E}_d = \mathcal{E}_\delta \cup \mathcal{E}_d \\ \min\{N_{x_0}(e), N_{x_\delta + d}(e)\} = \max\{N_{x_0}(e), N_{x_\delta + d}(e)\}, \quad \forall e \in \mathcal{E}_\delta \cap \mathcal{E}_d \end{cases}$$

$$\Rightarrow \quad \begin{cases} \varphi^n(\delta \oplus d) \leq \varphi^n(\delta \oplus d'), \quad \forall d' \in \mathcal{D}(x_\delta) \\ \mathcal{E}_\delta = \mathcal{E}_d \\ N_{x_0}(e) = N_{x_\delta + d}(e), \qquad \forall e \in \mathcal{E}_\delta \end{cases}$$

$$\Rightarrow \quad \begin{cases} \varphi^n(\delta \oplus d) \leq \varphi^n(\delta \oplus d'), \quad \forall d' \in \mathcal{D}(x_\delta) \\ c(x_0) = c(x_\delta + d) \end{cases}$$

$$\Rightarrow \quad \begin{cases} \varphi^n(\delta \oplus d) \leq \varphi^n(\delta \oplus d'), \quad \forall d' \in \mathcal{D}(x_\delta) \\ \varphi^c(\delta \oplus d) = 0. \end{cases}$$

Clearly, any policy $\delta' = \delta \oplus (d_{m+1}, \ldots, d_{m+q}) \in \hat{\underline{\Pi}}^{=}(x_0, \Phi^c, \Phi^n)$ with $d_{m+1} \neq d$ and $q \geq 1$ cannot dominate $\delta \oplus d$ because $\varphi^n(\delta') \geq \varphi^n(\delta \oplus d_{m+1}) \geq \varphi^n(\delta \oplus d)$ and $\varphi^c(\delta') \geq 0 = \varphi^c(\delta \oplus d)$. □

References

[1] Bard, J. F., Purnomo, H. W., 2005. Hospital-wide reactive scheduling of nurses with preference considerations. IIE Transactions 37 (7), 589–608.

[2] Bard, J. F., Purnomo, H. W., 2005. Short-term nurse scheduling in response to daily fluctuations in supply and demand. Health Care Management Science 8 (4), 315–324.

[3] Burke, E., De Causmaecker, P., Vanden Berghe, G., 2004. The state of the art of nurse rostering. Journal of Scheduling 7 (6), 441–499.

[4] Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D., 2004. An annotated bibliography of personnel scheduling and rostering. Annals of Operations Research 127 (1–4), 21–144.

[5] Froger, C., 2015. Mise à jour des horaires de personnel travaillant sur des quarts. Master's dissertation, Polytechnique Montréal. In French.

[6] Gross, C. N., Fügener, A., Brunner, J. O., 2018. Online rescheduling of physicians in hospitals. Flexible Services and Manufacturing Journal 30 (1), 296–328.

[7] Hassani, R., Desaulniers, G., Elhallaoui, I., 2017. Real-time personnel re-scheduling after a minor disruption. Technical report, Les Cahiers du GERAD G–2017–27, HEC Montréal, Montréal.

[8] Irnich, S., Desaulniers, G., 2005. Shortest path problems with resource constraints. In: Desaulniers, G., Desrosiers, J., Solomon, M. M. (Eds.), Column Generation. Springer US, Boston, MA, Ch. 2, pp. 33–65.

[9] Kitada, M., Morizawa, K., 2013. A heuristic method for nurse rerostering problem with a sudden absence for several consecutive days. International Journal of Emerging Technology and Advanced Engineering 3 (11), 353–361.

[10] Kitada, M., Morizawa, K., Nagasawa, H., 2010. A heuristic method in nurse rerostering following a sudden absence of nurses. In: Proc. 11st Asia Pacific Industrial Engineering & Management Systems Conference. Vol. 6.

[11] Mahenhout, B., Vanhoucke, M., 2011. An evolutionary approach for the nurse rerostering problem. Computers & Operations Research 38 (10), 1400–1411.

[12] Moz, M., Pato, M. V., 2003. An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals of Operations Research 119 (1–4), 285–301.

[13] Moz, M., Pato, M. V., 2004. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. Annals of Operations Research 128 (1–4), 179–197.

[14] Moz, M., Pato, M. V., 2007. A genetic algorithm approach to a nurse rerostering problem. Computers & Operations Research 34 (3), 667–691.

[15] Pato, M. V., Moz, M., 2008. Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. Journal of Heuristics 14 (4), 359–374.

[16] Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L., 2013. Personnel scheduling: A literature review. European Journal of Operational Research 226 (3), 367–385.