Les Cahiers du GERAD ISSN: 0711–2440

Parallel stimulation of disruptions for personnel scheduling in a flexible working environment

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

G–2021–01

January 2021

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, 2021
– Bibliothèque et Archives Canada, 2021

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, 2021
– Library and Archives Canada, 2021

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

Parallel stimulation of disruptions for personnel scheduling in a flexible working environment

Rachid Hassani

Guy Desaulniers

Issmail El Hallaoui

GERAD & Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Montréal (Québec), Canada H3C 3A7

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

January 2021
Les Cahiers du GERAD
G–2021–01

Abstract : Personnel scheduling aims to determine least-cost personnel schedules to meet the demand for employees in each period of a planning horizon. In this article, we propose a heuristic, called Parallel Stimulation of Disruptions Heuristic (PSDH), for solving a personnel scheduling problem. PSDH is a new, integrated approach for this type of problem, which generates and affects shifts simultaneously. It is based on the iterative stimulation/correction of a set of disruptions on certain employee schedules. Each disruption is targeted according to predetermined probabilistic improvement scores, and repaired using an algorithm inspired by the heuristic of Hassani et al. [15], which re-optimizes a schedule following a minor disruption. The approach is also based on a partition of the current solution, which is updated at each iteration to stimulate/repair the maximum number of disruptions in parallel. The proposed algorithm has been tested on real-life instances involving up to 94 employees and 10 jobs. PSDH found solutions of very good quality (2.01% from optimality on average) in fast computational times (less than three minutes on average).

Keywords: Personnel scheduling, parallel iterative heuristic, partitioning, disruption simulation

Acknowledgements: 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 problems are common in many areas such as retail, transport, healthcare and industrial production. There are many variations of the problem depending on the environment in which the work is performed. In the non-continuous environment, we work during the day with fixed opening and closing hours (e.g., in superstores), whereas in a continuous environment, work is carried out 24 hours a day (e.g., in hospitals, factories). Our focus of study in this paper is on retail, which can be seen as part of the non-continuous environment because most of the work is done during the day but some limited activities may also be conducted overnight. In retail, payroll expenses constitute the major operating costs for a company. Since the reduction of these costs has a direct impact on a retailer's ability to lower the price of goods, competition drives companies to develop new tools to reduce payroll costs. New strategies aimed at improving personnel scheduling can be powerful tools for reducing such costs.

Often, companies manage many employees with various qualifications. Producing an appropriate schedule that minimizes labor costs and meets demand, while respecting union and legislative constraints, can be a challenging task. For this reason, many employers are interested in optimization software. Consequently, many operations research projects are carried out with the goal of improving scheduling software. Since the exact resolution of these complex problems is very costly in terms of computing time, most of the proposed methods are heuristics, which sacrifice the guarantee of optimality in favor of producing a high-quality schedule in a reasonable amount of time. Research is increasingly focused on reducing this computational time while maintaining the quality of the schedule adopted in the end.

In this paper, we develop a heuristic, called Parallel Stimulation of Disruptions Heuristic (PSDH), that optimizes a personnel schedule. PSDH is inspired by the heuristic proposed by Hassani et al. [15] for the real-time re-optimization of a personnel schedule after a minor disruption. We place ourselves in a multi-job context with very heterogeneous personnel in terms of availability, qualifications, and authorized working hours per day. Our context is also characterized by a demand which can fluctuate at any time of the planning horizon. This means we can have shifts that can start and end at many different times during the same day.

1.1 Literature review

In the early 1950s, Edie [9] proposed a heuristic method to optimize the waiting time of vehicles arriving at New York toll booths. A few months later, Dantzig [7] proposed solving the same problem using a linear integer program, presenting a model capable of providing a personnel schedule to meet the needs in question at a lower cost. The Dantzig study has become a benchmark, and researchers have sought to improve this model in order, among other things, to adapt it to other personnel scheduling problems. For more background on the topic, several survey papers on personnel scheduling [4, 6, 10] can be consulted.

Personnel scheduling problems are combinatorial in nature and are often characterized by the presence of several conflicting constraints. To solve these complex problems, many researchers have developed metaheuristics, sacrificing the guarantee of optimality in exchange for a very good quality solution. Among those metaheuristics that have proven their efficiency, we find local search methods whose approach consists in iteratively improving just one solution at a time. These methods are based on two essential elements (i) a neighborhood structure and (ii) a procedure which induces a topology on this neighborhood. Simulated annealing [17], tabu search [13, 14], and variable neighborhood search [20] are the most popular local search methods.

Most works that have addressed personnel scheduling problems are associated with the hospital domain (for example, scheduling nurses [6, 18]). This area has a very limited number of possible shifts (for example, 7am to 3pm, 3pm to 11pm, and 11pm to 7am). Demand is often expressed by the number of shifts required for a job. In addition, this domain is generally characterized by the presence

of several different horizontal constraints (for example, on shift type sequences or days off patterns), which makes it less flexible than the retail domain but still very challenging.

For personnel scheduling problems, we distinguish two types of approaches proposed in the literature (i) explicit approaches, which consist in choosing the shifts retained among sets of enumerated candidate shifts [1, 9, 11, 19, 24] and (ii) the implicit approaches wherein potential shifts are not enumerated a priori but rather built during the solution process [2, 12, 16, 21, 23]. In a context similar to ours, in 2004, Musliu et al. [22] proposed an implicit tabu search method in order to choose a set of shifts among four types of shifts (each type is characterized by a start time window and a duration window), and to determine the number of employees needed for each shift. In this study, the neighborhood is determined by the following moves: add/remove a shift, change the start/duration of a shift, and change the number of employees needed for a shift type on a given day. Compound moves are also used. To avoid local minimums, two tabu lists are used: one in which we record the inverse of the moves performed, the other in which we record the solutions found. Note that the final assignment of employees to shifts, depending on their availability and qualifications, is not considered in their context (unlike ours). The same problem was studied in later works [3, 8, 25].

In 2017, Bonutti et al. [5] addressed a variant of the problem considered by Musliu et al. [22] that involves several jobs (namely, 2 or 3 jobs) and new shift types containing breaks. They proposed a local search method based on simulated annealing. The neighborhood was defined using moves like those defined above. But here they considered two types of neighborhoods, the first of which is formed using all the basic moves, except for one move with which the second neighborhood is defined. It turned out that the sequential search on these two neighborhoods worked better than the classical neighborhood search using all the moves at the same time.

1.2 Discussions and contributions

From the literature review, we can make the following observations on personnel scheduling in the retail field:

- The retail field is rarely discussed in the literature;
- When the personnel are heterogeneous, only explicit approaches are used. Indeed, all of the implicit approaches proposed aim to find an anonymous schedule in which the characteristics of the employees are omitted. When these characteristics are very different, the assignment of anonymous shifts to employees is not obvious.

In this paper, we discuss the global optimization of a retail personnel schedule by introducing PSDH, an integrated method that generates and assigns shifts simultaneously during resolution. We place ourselves in a context which is characterized by a demand which can fluctuate at any time of the given horizon, with very heterogeneous personnel in terms of availability, qualifications, and authorized working hours per day. On the one hand, this context offers a certain flexibility for the construction of the schedule. On the other hand, this flexibility leads to very combinatorial problems that are difficult to solve. To overcome these problems, we develop an algorithm that stimulates minor disruptions in the current solution and then repairs them using an algorithm inspired by that of Hassani et al. [15], which allows the correction of only one disruption at a time. The main design challenges consist in the choice of the disruptions to be stimulated and in the partitioning of the current solution, which is updated dynamically, and which must be considered during the resolution in order to deal with the maximum number of disruptions in parallel.

1.3 Paper structure

This paper is organized as follows. In Section 2, we define notions that are used subsequently, and we formulate the problem. In Section 3, we start by introducing terminology and concepts that will be useful for us in describing PSDH. Thereafter, we detail this heuristic as well as the procedures on which it relies. In Section 4, we present and analyze the results of our numerical experiments. Finally, conclusions are drawn in Section 5.

2 Problem formulation

In this section, we start by stating some general concepts and then we formulate the personnel scheduling problem that must be solved to find optimal employee schedules. Finally, we present an integer linear programming model for this problem.

2.1 General notions

A few days in advance (or even a few weeks), the employer must be able to inform his employees about their working days and days off, as well as the jobs they will have to perform for a given horizon (generally one week). The employer, in planning the schedule, seeks to minimize the company's expenses related to the payroll while considering the following:

- *Satisfaction of demand*: Faced with customer demand, the company must be able to respond to customers with a reasonable number of employees. If there are too few of them, the demand will obviously not be met. Having too many employees is likewise suboptimal, because the company can a priori mobilize less, while maintaining its quality of service. Either way, it is losing money. In our study, we consider that the number of clients to be served can be equivalently expressed as the number of employees required for the service. Consequently, the demand will hereafter be expressed by the number of employees required.
- *Working hours constraints*: Unions as well as labor laws impose a number of constraints related to employee schedules. Among these, we find a minimum and maximum duration for a shift (i.e., an employee's working time for one day); a maximum working time for the week; a minimum rest time between two shifts; and a minimum number of rest days per week.
- *Employee qualifications*: Each employee is qualified to perform a subset of the jobs. The employer will not assign a job to an employee who does not have the necessary qualifications.
- *Employee availability*: The availability of each employee may vary from day to day. If a shift overlaps with an employee's downtime, then it cannot be assigned to that employee.

We are interested in the basic planning problem described by Hassani et al. [15]. It consists in finding an optimal (or near-optimal) work schedule for a set of employees $\mathcal{E}$ and for a set of jobs $\mathcal{W}$, over a 7-day horizon $\mathcal{H} = \{1, \ldots, 7\}$. The horizon is discretized in periods of 15 minutes (a different discretization could be used). Let $\mathcal{P}$ be the set of these periods. We denote by $\mathcal{P}^h$ the restriction of set $\mathcal{P}$ to day $h \in \mathcal{H}$. Subsequently, we characterize a shift s by its start period $b_s \in \mathcal{P}$, its end period $f_s \in \mathcal{P}$, and its length in periods $l_s (= f_s - b_s + 1)$. We also assume that this shift is associated with a single job $w_s \in \mathcal{W}$. The demand, expressed in the number of employees required for job $w \in \mathcal{W}$ in the period $p \in \mathcal{P}$, is given by d_p^w. Each employee $e \in \mathcal{E}$ is qualified for a set of jobs $\mathcal{W}_e \subseteq \mathcal{W}$ and available for periods $\mathcal{P}_e \subseteq \mathcal{P}$. We denote by $\mathcal{P}_e^h$ the subset of periods in $\mathcal{P}_e$ of the day $h \in \mathcal{H}$. We denote by $\mathcal{S}_h^e$ the set of shifts that can be proposed for employee $e \in \mathcal{E}$ during day $h \in \mathcal{H}$. A shift s belongs to $\mathcal{S}_h^e$ if (i) $b_s \in \mathcal{P}_h$, (ii) $w_s \in \mathcal{W}_e$, (iii) $\{b_s, \ldots, f_s\} \subset \mathcal{P}_e$, and (iv) $\underline{l}_e \leqslant l_s \leqslant \bar{l}_e$, where $\underline{l}_e$ (resp. $\bar{l}_e$) is the minimum (resp. maximum) duration for which the employee e can work continuously during a day. Under-coverage (having fewer employees than demand at a given period) is prohibited and over-coverage (having more employees than demand at a given period) is penalized.

Under these conditions, there is a risk of not finding a feasible work schedule if there are not enough personnel available to cover the entire demand. To avoid this infeasibility, the use of anonymous shifts is permitted. Anonymous shifts are shifts that are not assigned to any employee but are counted as shifts performed to meet demand. Usually, these shifts are assigned to employees from another department, or to temporary employees just prior to operations. We denote by $\mathcal{S}^A$ the complete set of anonymous shifts and by $\mathcal{S}_h^A$ the subset of anonymous shifts proposed during day $h \in \mathcal{H}$. The length of each anonymous shift must be between two limits, $\underline{l}^A$ and $\bar{l}^A$.

The employer's goal is not only to minimize labor costs, but also to avoid penalties for anonymous shifts and over-coverage. All these costs are not necessarily real costs incurred by the company, but they correspond to a structure penalizing those situations we wish to avoid in the schedule. They

also ensure equity between employees in terms of the number of hours worked over the time horizon. Indeed, there is a strictly increasing function on the remuneration levels $\mathcal{K}^L = \{1, 2, \ldots, |\mathcal{K}^L|\}$. Each level is made up of 8 hours of work (a different duration could be used). For example, if an employee e works 20 hours, then the first 8 hours will be paid up to c_0 per period, the second 8 hours up to τc_0 per period (where τ is the growth rate), and finally the last 4 hours will be paid up to $\tau^2 c_0$ per period. We say, then, that the remuneration of e reaches level 3. The remuneration at level $k \in \mathcal{K}^L$ and the number of periods on this level are denoted c_k^L and n_k^L, respectively.

The penalty incurred by using an anonymous shift $s \in \mathcal{S}^A$ is proportional to the shift length and is given by $c^Y l_s$, where c^Y is a unit penalty large enough to favor covering the demand with regular shifts. Over-coverage penalties are defined to ensure that over-coverage is spread across multiple periods whenever possible. Indeed, it is usually better to have, e.g., an extra employee at two different periods than two extra employees at the same period. Consequently, the over-coverage penalties also follow a strictly increasing step-wise function defined by a set of levels $\mathcal{K}^V = \{1, 2, \ldots, |\mathcal{K}^V|\}$ and a penalty rate c_k^V per employee on level $k \in \mathcal{K}^V$. The number of periods on level $k \in \mathcal{K}^V$ is denoted n_k^V.

If we assume that we have the sets of anonymous shifts proposed, as well as the sets of shifts proposed for each employee $e \in \mathcal{E}$, then we can view the desired schedule as a solution of an integer linear program (ILP). The decision variables of this program make it possible to select (i) for each employee, certain shifts among his personalized shifts and (ii) a set of anonymous shifts among those proposed. If these sets are constructed in an exact way (i.e., all feasible shifts are considered), then the established schedule is an optimal schedule. Otherwise, these sets are generated heuristically and the returned schedule represents a sub-optimal schedule for the real planning problem. In this case, the quality of the schedule is strongly dependent on the efficiency of the heuristic used to generate the potential shifts.

2.2 Mathematical model

The proposed ILP also uses the following notation:

Constants	
n^R:	Minimum duration, in periods, of rest between two consecutive shifts assigned to the same employee;
n^O:	Minimum number of days off for each employee;
a_{sw}^p:	Binary parameter equal to 1 if shift $s \in \mathcal{S}$ covers job $w \in \mathcal{W}$ in period $p \in \mathcal{P}$, 0 otherwise;
Constants	
X_{es}^h:	Binary variable equal to 1 if we assign shift $s \in \mathcal{S}_h^e$ to employee $e \in \mathcal{E}$ on day $h \in \mathcal{H}$, 0 otherwise;
O_e^h:	Binary variable which indicates the assignment of a day off to employee $e \in \mathcal{E}$ on day $h \in \mathcal{H}$;
Y_s:	Integer variable which counts the number of times anonymous shift $s \in \mathcal{S}^A$ is used;
L_e^k:	Integer variable which gives the number of periods worked by employee $e \in \mathcal{E}$, on level $k \in \mathcal{K}^L$;
V_w^{kp}:	Integer variable which specifies the number of over-coverage for job $w \in \mathcal{W}$ ìn period p on level $k \in \mathcal{K}^V$;

Given this notation, the personnel scheduling problem considered can be formulated as the following ILP:

$$\text{Minimize} \quad \sum_{e\in\mathcal{E}} \sum_{k\in\mathcal{K}^L} c_k^L L_e^k + \sum_{s\in\mathcal{S}^A} c^Y l_s Y_s + \sum_{p\in\mathcal{P}} \sum_{w\in\mathcal{W}} \sum_{k\in\mathcal{K}^V} c_k^V V_w^{kp} \tag{1}$$

$$\text{subject to:} \quad \sum_{s\in\mathcal{S}_h^e} X_{es}^h + O_e^h = 1, \qquad \forall e \in \mathcal{E}, h \in \mathcal{H} \tag{2}$$

$$\sum_{h\in H} O_e^h \geqslant n^O, \qquad \forall e \in \mathcal{E} \tag{3}$$

$$\sum_{h\in\mathcal{H}} \sum_{s\in\mathcal{S}_h^e} l_s X_{es}^h - \sum_{k\in\mathcal{K}^L} L_e^k = 0, \qquad \forall e \in \mathcal{E} \tag{4}$$

$$M\, O_e^{h+1} + \sum_{s\in\mathcal{S}_{h+1}^e} b_s X_{es}^{h+1} - \sum_{s\in\mathcal{S}_h^e} (f_s + 1 + n^R) X_{es}^h \geqslant 0, \qquad \forall e \in \mathcal{E}, h \in \mathcal{H}\backslash\{7\} \tag{5}$$

$$\sum_{e\in\mathcal{E}}\sum_{h\in\mathcal{H}}\sum_{s\in\mathcal{S}_h^e} a_{sw}^p X_{es}^h + \sum_{s\in\mathcal{S}^A} a_{sw}^p Y_s - \sum_{k\in\mathcal{K}^V} V_w^{kp} = d_w^p, \qquad \forall p\in\mathcal{P}, w\in\mathcal{W} \tag{6}$$

$$X_{es}^h \in \{0,1\}, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H}, s\in\mathcal{S}_h^e \tag{7}$$

$$O_e^h \in \{0,1\}, \qquad \forall e\in\mathcal{E}, h\in\mathcal{H} \tag{8}$$

$$L_e^k \in [0, n_k^L], \text{integer}, \qquad \forall e\in\mathcal{E}, k\in\mathcal{K}^L \tag{9}$$

$$Y_s \geqslant 0, \text{integer}, \qquad \forall s\in\mathcal{S}^A \tag{10}$$

$$V_w^{kp} \in [0, n_k^V], \text{integer}, \qquad \forall w\in\mathcal{W}, p\in\mathcal{P}, k\in\mathcal{K}^V. \tag{11}$$

The objective function (1) minimizes the sum of the labor costs and the penalties incurred by the anonymous shifts and the over-coverage. Constraints (2) ensure that each employee is assigned to a shift or to a day off for each day in horizon $\mathcal{H}$. Constraints (3) impose a minimum of n^O days off for each employee. The distribution of the work periods for each employee on the different levels of $\mathcal{K}^L$ is calculated by the constraints (4). Note that the relationships between c_k^L costs ensure that the cheapest levels are always used first. The constraints (5) enforce a rest of at least n^R periods between shifts assigned to the same employee on two consecutive days. For each job and each period, the constraints (6) guarantee that the demand is covered by a sufficient number of employees or anonymous shifts. They also make it possible to calculate the amount of over-coverage on each level of $\mathcal{K}^V$. Finally, the domains of the decision variables are limited by (7)–(11).

Let $\mathcal{X}$ be the set of feasible schedules for this problem. For a feasible schedule $x\in\mathcal{X}$, we denote by $\mathcal{S}_x^A$, all the anonymous shifts present in schedule x. Furthermore, $\mathcal{S}_x(e,h)$ represents the shift reserved for employee $e\in\mathcal{E}$ on day $h\in\mathcal{H}$. By misuse of language, we say that an employee is assigned to a zero shift, noted $s_0(h)$, when the employee is off on day $h\in\mathcal{H}$. In this case, we have $\mathcal{S}_x(e,h) = s_0(h)$. Finally, we consider the following two functions.

$\mathcal{V}(e,[d_1,d_2])$ is equal to 1 if employee $e\in\mathcal{E}$ is available between periods d_1 and d_2 and can work for the duration d_2-d_1+1, and to 0 otherwise;

$\mathcal{Q}(e,w)$ is equal to 1 if employee $e\in\mathcal{E}$ is qualified for job $w\in\mathcal{W}$, and to 0 otherwise.

Thus, a shift s can be assigned to employee e if $\mathcal{A}(e,s) := \mathcal{V}(e,[b_s,f_s]) \times \mathcal{Q}(e,w_s) = 1$.

3 Solution algorithm

PSDH is an integrated algorithm which generates/assigns shifts simultaneously for each specific employee during resolution. It is based on basic moves, called generating decisions, which generate new feasible/infeasible solutions from a given solution. It iteratively stimulates/corrects a set of well-targeted disruptions according to certain probabilistic scores, guiding PSDH to perform an effective search. Two clustering procedures are used to generate a set of subproblems which will be dealt with in parallel. On each subproblem, we stimulate a disruption that leads us, generally, to an infeasible solution. So, we define a neighborhood on this subproblem, whose size is determined by a bound on the number of modifications required to correct the disruption and return to the feasible space.

In the first part of this section, we introduce the terminology and concepts that are useful for describing and formalizing our procedures. Next, we describe the clustering algorithms used before presenting the scores on which we base our choice for the type of disruption to be stimulated in each subproblem. Finally, we detail the proposed heuristic.

3.1 Concepts and terminology

In order to move around the solution space $\mathcal{X}$, we introduce the following types of generating decisions g^S, g^X, g^{SA} and g^{XA}:

$g^S(h,e_1,e_2)$: Swaps shift assignment for employees e_1 and e_2 during day h;

$g^X(h,e_1,e_2,l)$: Extends a shift of e_1 so as to cover the first or last l shift periods of e_2 during day h;

$g^{SA}(h, e, s)$**:** Assigns the anonymous shift s to employee e during day h, and replaces the current shift of employee e, if different from $s_0(h)$, with an anonymous shift;

$g^{XA}(h, e, s, l)$**:** Consists in extending the shift of e so as to cover the l first or last periods of the anonymous shift s during day h.

The application of a generating decision may lead to an infeasible solution (for example, for $g^{SA}(h, e, s)$ if $\mathcal{A}(e, s) = 0$). This will imply the need to apply a sequence of generating decisions $(g_1, g_2, \ldots, g_m)$ to obtain a feasible solution. When the sequence $(g_1, g_2, \ldots, g_m)$ is minimal, in the sense that each subsequence $(g_1, g_2, \ldots, g_n)$ with $n < m$ leads to an infeasible solution, we say that the sequence $(g_1, g_2, \ldots, g_m)$ represents an elementary decision. Let n^g be the maximum number of generating decisions that can form an elementary decision.

Algebraically, the application of an elementary decision $(g_1, g_2, \ldots, g_m)$ can be seen as a transition vector d between a feasible solution $x \in \mathcal{X}$ and another feasible solution $y \in \mathcal{X}$, i.e., $d = y - x$. We denote by $\mathcal{D}(x)$ the set of all elementary decisions admissible from solution $x \in \mathcal{X}$. Each decision $d \in \mathcal{D}(x)$ generates a number of modifications n_d and a cost c_d that can be negative if the decision improves the current solution or non-positive otherwise. Note that we define a modification as any change to an employee's schedule or an anonymous shift after making a sequence of decisions. When this sequence brings several changes to the schedule of the same employee, or to the same anonymous shift on the same day, these changes are counted as a single modification. In the following, any sequence δ of elementary decisions is called a policy. If $\delta = (d_1, d_2, \ldots, d_m)$, we say that δ is a policy of size m. The application of this policy generates a sequence of feasible solutions $\{x_i = x_{i-1} + d_i\}_{i=1}^{m}$. In this case, we characterize policy δ by cost $\varphi^c(\delta) = \sum_{i=1}^{m} c_{d_i}$ and by the number of modifications $\varphi^n(\delta)$ (the modifications made to the initial solution x_0 in order to obtain the solution x_m). Let $\Pi(x)$ be the set of admissible policies from $x \in \mathcal{X}$. We denote by $\Pi(x, \Phi^n) = \{\delta \in \Pi(x) | \varphi^n(\delta) \leqslant \Phi^n\}$ the subset of policies such that the number of changes generated is less than Φ^n. Let $\mathcal{E}_\delta$ (resp. $\mathcal{S}_\delta$) be the set of employees (resp. anonymous shifts) involved in policy δ.

Let $x \in \mathcal{X}$ be a feasible solution, $n_x^A(h)$ the number of anonymous shifts on day $h \in \mathcal{H}$, and $\mathcal{E}_x(k) \subset \mathcal{E}$ the set of employees such that their remuneration reaches level $k \in \mathcal{K}^L$. We denote by $n_x(k)$ the cardinality of this set. For each employee $e \in \mathcal{E}$, we denote by $p_x(e)$ (resp. $r_x(e)$) the total number of periods programmed for employee e (resp. the number of days off) in schedule x. With these notations, we define the vectors $N_x^A = (n_x^A(h))_{h \in \mathcal{H}}$, $N_x = (n_x(k))_{k \in \mathcal{K}^L}$, $P_x(\mathcal{E}') = (p_x(e))_{e \in \mathcal{E}'}$, and $R_x(\mathcal{E}') = (r_x(e))_{e \in \mathcal{E}'}$ where $\mathcal{E}' \subset \mathcal{E}$.

For each day $h \in \mathcal{H}$, we define the function $dist_h$ which will help us "measure" the flexibility of changes which may be made between the schedules of two employees:

$$\begin{aligned} dist_h : \mathcal{E} \times \mathcal{E} &\longrightarrow [0, 1] \\ (e_1, e_2) &\longmapsto \left(1 - \frac{|\mathcal{P}_{e_1}^h \cap \mathcal{P}_{e_2}^h|}{|\mathcal{P}_{e_1}^h \cup \mathcal{P}_{e_2}^h|}\right)^{|\mathcal{W}_{e_1} \cap \mathcal{W}_{e_2}|}. \end{aligned}$$

The term $1 - \frac{|\mathcal{P}_{e_1}^h \cap \mathcal{P}_{e_2}^h|}{|\mathcal{P}_{e_1}^h \cup \mathcal{P}_{e_2}^h|}$ measures the percentage of overlap between subsets $\mathcal{P}_{e_1}^h$ and $\mathcal{P}_{e_2}^h$ for two employees e_1 and e_2 during day h. We choose to weight this percentage according to the number of jobs for which the two employees e_1 and e_2 are qualified. In addition, this score depends only on the initial data of the problem and not on a particular solution. Thereafter, for a given vector $q = (q_i)_{1 \leqslant i \leqslant n}$ where $q_i \in [a, b]$, we define $var(q)$:

$$var(q) = \frac{\frac{1}{n}\sum_{i=1}^{n} q_i^2 - (\frac{1}{n}\sum_{i=1}^{n} q_i)^2}{b^2 - a^2}.$$

The closer $var(q)$ is to 1, the greater the dispersion of the q_i values. Thereafter, we use $var(N_x^A)$, $var(N_x)$, $var(P_x)$, $var(R_x)$ and $dist_h$ to define clusters for the problem and choose the type of disruption to stimulate in each subproblem considered.

We call $(\mathcal{C}_1(x), \mathcal{C}_2(x), \ldots, \mathcal{C}_\zeta(x))$ a clustering of the problem from solution x into ζ subproblems if $\mathcal{C}_i(x) = (\mathcal{E}_i, \mathcal{S}_i^A(x)), \forall i \in \{1, \ldots, \zeta\}$, are such that:

$$\begin{cases} \bigcup_{i=1}^{\zeta} \mathcal{E}_i = \mathcal{E} \\ \mathcal{E}_i \bigcap \mathcal{E}_j = \varnothing, \qquad \forall i \neq j \\ \bigcup_{i=1}^{\zeta} \mathcal{S}_i^A(x) = \mathcal{S}^A(x) \\ \mathcal{S}_i^A(x) \bigcap \mathcal{S}_j^A(x) = \varnothing, \qquad \forall i \neq j. \end{cases}$$

For each subproblem $\mathcal{C}(x)$, we denote by $\Pi(x, \phi^n, \mathcal{C}(x))$ the subset of policies of $\Pi(x, \Phi^n)$ which only involve employees and anonymous shifts present in the subproblem $\mathcal{C}(x)$. If $\delta \in \Pi(x, \Phi^n, \mathcal{C}(x))$ and $\delta^{'} \in \Pi(x, \Phi^n, \mathcal{C}^{'}(x))$ where $\mathcal{C}(x)$ and $\mathcal{C}^{'}(x)$ are two subproblems not necessarily coming from the same clustering, we say that the two policies are compatible if $\mathcal{E}_\delta \cap \mathcal{E}_{\delta'} = \varnothing$ and $\mathcal{S}_\delta^A \cap \mathcal{S}_{\delta'}^A = \varnothing$. In this case we have $\delta \oplus \delta^{'} \in \Pi(x)$, where the symbol $\oplus$ indicates the concatenation of policies δ and $\delta^{'}$. In essence, all policies associated with subproblems of the same clustering are compatible.

We consider a subproblem $\bar{\mathcal{C}}(x) = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$. The application of a generating decision g^Y, with $Y \in \{S, X, SA, XA\}$, $\mathcal{E}_{g^Y} \subset \bar{\mathcal{E}}$ and $\mathcal{S}_{g^Y}^A \subset \bar{\mathcal{S}}^A(x)$, can be seen as a stimulation of disruption in the subproblem $\bar{\mathcal{C}}(x)$. If this generating decision leads us to an infeasible solution, then this disruption requires a correction by applying other generating decisions to arrive at a feasible solution. If not, then that decision, in itself, is an elementary decision. When a generating decision g^Y of type $Y \in \{S, X, SA, XA\}$ acts as a disruption stimulant, we use the following notation: $\widehat{g}^Y$. In order to find a policy $\delta \in \Pi(x, \Phi^n, \bar{\mathcal{C}}(x))$ formed by m elementary decisions, we need to stimulate at least m disruptions. Moreover, if the cost of this policy $\varphi^c(\delta)$ is negative, then we say that policy δ is improving. Subsequently, we denote by $\bar{\Pi}(x, \Phi^n, \bar{\mathcal{C}}(x))$ (resp. $\bar{\Pi}(x)$) the subset of improving policies of $\Pi(x, \Phi^n, \bar{\mathcal{C}}(x))$ (resp. $\Pi(x)$).

Let $\bar{\delta}(x) \in \arg\min_{\delta \in \bar{\Pi}(x)} \varphi^c(\delta)$. It is easy to see that the solution $x + \bar{\delta}(x) \in \mathcal{X}$ is an optimal solution of the problem. Our goal then is to find or approximate policy $\bar{\delta}(x)$. For this, we propose using policies in $\bar{\Pi}(x, \Phi^n, \mathcal{C}_i(x))$, $i \in \{1, 2, \ldots, \zeta\}$, where ζ is the number of subproblems considered. The challenge is to identify subproblems on which to stimulate and correct disruptions $\widehat{g}^Y$ such that the resulting solutions yield a cost improvement. The choice of each type of disruption is based on scores that emulate the probability that the disruption $\widehat{g}^Y$ leads us to a policy of $\bar{\Pi}(x, \Phi^n, \bar{\mathcal{C}}(x))$. The subproblems considered and their number are dynamically updated during the resolution according to certain quantities (defined subsequently). The notion of compatibility between policies allows us to use parallel programming; the subproblems considered are dealt with simultaneously.

3.2 Clustering

To build the subproblems that will be later used in PSDH, we need to group together some employees and some anonymous shifts in a given feasible schedule. To do so, we propose two clustering algorithms called `clusteringE` and `clusteringA`.

In `clusteringE` (Algorithm 1), we start with the partitioning of the set of employees $\mathcal{E}$ (Steps 4–8) so that each set $\mathcal{E}_i$ contains the minimum number of employees of $\mathcal{E}_x(k)$ for each level $k \in \mathcal{K}^L$. This allows us to have $\mathcal{E}_i$ sets such that $var(P_x(\mathcal{E}_i))$ is quite significant. Indeed, the set $\mathcal{E}_i$ contains employees working too much in schedule x, and others working too less. Then, for each anonymous shift $s \in \mathcal{S}^A(x)$ and set $\mathcal{E}_i$, we calculate a score $\mu_i(s)$ (Step 13) which approximates the "probability" that this anonymous shift be worked, totally or partially, by the employees of $\mathcal{E}_i$. This score takes into account the qualifications and availability of the employees in question. Note that the term associated with qualifications, namely n_q, is more heavily weighted because even if an employee is unavailable to work for the entire anonymous shift s (i.e., $\mathcal{V}(e, [b_s, f_s]) = 0$), he can work a partial shift (for example,

using the decision g^{XA}). Finally, the anonymous shift s is assigned to the subproblem with the largest score.

Algorithm 1 : `clusteringE`

input : initial schedule x;
number of subproblems ζ ;
output : subproblems $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\zeta)$;

1 **for** $i \in \{1, \ldots, \zeta\}$ **do**
2 $\quad \mathcal{E}_i \leftarrow \varnothing$; $\mathcal{S}_i^A(x) \leftarrow \varnothing$;
3 $i \leftarrow 1$;
4 **foreach** $k \in \mathcal{K}^L$ **do**
5 $\quad$ **foreach** $e \in \mathcal{E}_x(k)$ **do**
6 $\quad\quad \mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{e\}$;
7 $\quad\quad$ **if** $i < \zeta$ **then** $i \leftarrow i + 1$;
8 $\quad\quad$ **else** $i \leftarrow 1$;
9 **foreach** $s \in \mathcal{S}^A(x)$ **do**
10 $\quad$ **for** $i \in \{1, \ldots, \zeta\}$ **do**
11 $\quad\quad n_q \leftarrow \frac{|\{e \in \mathcal{E}_i | \mathcal{Q}(e, w_s) = 1\}|}{|\mathcal{E}_i|}$;
12 $\quad\quad n_d \leftarrow \frac{|\{e \in \mathcal{E}_i | \mathcal{V}(e, [b_s, f_s]) = 1\}|}{|\mathcal{E}_i|}$;
13 $\quad\quad \mu_i(s) \leftarrow n_d^2\, n_q$;
14 $\quad i_0 \leftarrow \arg \max_{i \in \{1, \ldots, \zeta\}} \mu_i(s)$;
15 $\quad \mathcal{S}_{i_0}^A(x) \leftarrow \mathcal{S}_{i_0}^A(x) \cup \{s\}$;
16 **for** $i \in \{1, \ldots, \zeta\}$ **do**
17 $\quad \mathcal{C}_i \leftarrow (\mathcal{E}_i, \mathcal{S}_i^A(x))$;
18 return $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\zeta)$;

Unlike `clusteringE`, `clusteringA` (Algorithm 2) begins by forming the sets $\mathcal{S}^A(x)$. Then, we assign employees to the subproblem that contains the most anonymous shifts likely to be worked by the employee in question (Step 12). We denote by $\mathcal{C}^E(x, \zeta)$ (resp. $\mathcal{C}^A(x, \zeta)$) the set of subproblems returned by `clusteringE` (resp. `clusteringA`).

Algorithm 2 : `clusteringA`

input : initial schedule x;
number of subproblems ζ ;
output : subproblems $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\zeta)$;

1 **for** $i \in \{1, \ldots, \zeta\}$ **do**
2 $\quad \mathcal{E}_i \leftarrow \varnothing$; $\mathcal{S}_i^A(x) \leftarrow \varnothing$;
3 $i \leftarrow 1$;
4 **foreach** $s \in \mathcal{S}_x^A$ **do**
5 $\quad \mathcal{S}_i^A(x) \leftarrow \mathcal{S}_i^A(x) \cup \{s\}$;
6 $\quad$ **if** $i < \zeta$ **then** $i \leftarrow i + 1$;
7 $\quad$ **else** $i \leftarrow 1$;
8 **foreach** $e \in \mathcal{E}$ **do**
9 $\quad$ **for** $i \in \{1, \ldots, \zeta\}$ **do**
10 $\quad\quad \rho_i(e) \leftarrow |\{s \in \mathcal{S}_i^A(x) | \mathcal{A}(e, s) = 1\}|$;
11 $\quad i_0 \leftarrow \arg \max_{i \in \{1, \ldots, \zeta\}} \rho_i(s)$;
12 $\quad \mathcal{E}_{i_0} \leftarrow \mathcal{E}_{i_0} \cup \{e\}$;
13 **for** $i \in \{1, \ldots, \zeta\}$ **do**
14 $\quad \mathcal{C}_i \leftarrow (\mathcal{E}_i, \mathcal{S}_i^A(x))$;
15 return $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\zeta)$;

3.3 Disruption stimulation

Once the problem has been partitioned into subproblems by Algorithms 1 and 2, disruptions must be stimulated in each subproblem. Let $x \in \mathcal{X}$ and $\zeta \in \mathbb{N}^*$. We consider a subproblem $\bar{\mathcal{C}}(x) = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$

of $\mathcal{C}^E(x,\zeta) \cup \mathcal{C}^A(x,\zeta)$. Since we are only interested in improving policies, we only stimulate generating decisions likely to generate elementary decisions with a "very" negative cost. Stimulating several disruptions of different types and choosing the best one, that of the most negative cost, seems very time-consuming. Therefore, for our purposes, we define an order in which we stimulate, as quickly as possible, a disruption of a certain type $Y \in \{S, X, SA, XA\}$. For example, the order (SA, XA, S, X) indicates that we try to stimulate a disruption of type g^{SA} first. If this is not possible, we will try to stimulate a disruption of type g^{XA}, and so on. To find an order which maximizes the chance of quickly stimulating a "good" disruption, we define the following scores ν^S, ν^X, ν^{SA} and ν^{XA}:

$$\nu^S(\bar{\mathcal{C}}(x)) = \left[var(P_x(\bar{\mathcal{E}}))\right]^{1-var(R_x(\bar{\mathcal{E}}))} \tag{12}$$

$$\nu^X(\bar{\mathcal{C}}(x)) = \left[var(P_x(\bar{\mathcal{E}}))\right]^{var(R_x(\bar{\mathcal{E}}))} \tag{13}$$

$$\nu^{SA}(\bar{\mathcal{C}}(x)) = \sqrt[|\bar{\mathcal{S}}^A(x)|]{\frac{\sum_{e\in\bar{\mathcal{E}}} r_x(e)}{|\mathcal{H}||\bar{\mathcal{E}}|}} \tag{14}$$

$$\nu^{XA}(\bar{\mathcal{C}}(x)) = \sqrt[|\bar{\mathcal{S}}^A(x)|]{1 - \frac{\sum_{e\in\bar{\mathcal{E}}} r_x(e)}{|\mathcal{H}||\bar{\mathcal{E}}|}}. \tag{15}$$

The score $\nu^Y(\bar{\mathcal{C}}(x))$, $Y \in \{S, X, SA, XA\}$, emulates the probability that the stimulation of disruption $\widehat{g}^Y$ improves the current solution by obtaining an elementary decision of $\mathcal{D}(x)$ with a negative cost. In other words, $\nu^Y(\bar{\mathcal{C}}(x))$ allows us to see if the subproblem $\bar{\mathcal{C}}(x)$ represents a good basis for stimulating disruptions of type $Y \in \{S, X, SA, XA\}$ which will subsequently allow us to generate improving policies.

The formulas (12)–(15) are inspired from the following observations. Denote by $\bar{\mathcal{E}}^-$ (resp. $\bar{\mathcal{E}}^+$) all the employees of $\bar{\mathcal{E}}$ who work less (resp. more) than the average employee working time $\bar{p}_x = \frac{\sum_{e\in\bar{\mathcal{E}}} p_x(e)}{|\bar{\mathcal{E}}|}$. If $var(P_x(\bar{\mathcal{E}}))$ is high (i.e., the variability of values $p_x(e)$, $e \in \bar{\mathcal{E}}$, is significant), we will have a better chance of reducing the cost with g^S and g^X, by making the employees of $\mathcal{E}^-$ (resp. $\mathcal{E}^+$) work more (resp. less). When $var(R_x(\bar{\mathcal{E}}))$ is high, then we have employees who have a lot of days off and others who have less. In this case, the disruption stimulation using g^S may eventually lead us to elementary decisions $\mathcal{D}(x)$ with negative cost. If $var(R_x(\bar{\mathcal{E}}))$ is low, it would imply that the assignment of rest days is relatively homogeneous among employees of $\bar{\mathcal{E}}$. In this case, $\nu^X(\bar{\mathcal{C}}(x))$ favors the generating decisions g^X. On the other hand, when $|\bar{\mathcal{S}}^A(x)|$ is high, then the stimulation of a disruption using g^{SA} and g^{XA} looks interesting. Indeed, by nature, these two decisions reduce the number of periods covered by anonymous shifts, which implies a reduction in the cost of the current solution. In order to choose between g^{SA} and g^{XA}, we consider the value $\frac{\sum_{e\in\bar{\mathcal{E}}} r_x(e)}{|\mathcal{H}||\bar{\mathcal{E}}|}$, which represents the attendance rate on rest days for employees of $\bar{\mathcal{E}}$ over horizon $\mathcal{H}$. If this value is high then we favor the decision g^{SA}, in order to remove an anonymous shift and assign it to an employee who is supposed to be off duty in solution x. Otherwise, if $1 - \frac{\sum_{e\in\bar{\mathcal{E}}} r_x(e)}{|\mathcal{H}||\bar{\mathcal{E}}|}$ is high, then decision g^{XA} is favored. Indeed, in this case, we have several shifts assigned to employees that can be modified to shorten anonymous shifts $\bar{\mathcal{S}}^A(x)$.

Nonetheless, to determine the order in which we simulate a disruption, we do not compare the scores ν^S, ν^X, ν^{SA} and ν^{XA}. Instead, we consider the default order (SA, XA, S, X) and move to the end of this sequence any stimulation type $Y \in \{S, X, SA, XA\}$ such that $\nu^Y(\bar{\mathcal{C}}(x)) < \nu_0^Y$, where ν_0^Y is a predefined parameter value in $]0, 1[$. For example, if $\nu^{SA}(\bar{\mathcal{C}}(x)) < \nu_0^{SA}$, $\nu^{XA}(\bar{\mathcal{C}}(x)) \geqslant \nu_0^{XA}$, $\nu^S(\bar{\mathcal{C}}(x)) < \nu_0^S$ and $\nu^X(\bar{\mathcal{C}}(x)) \geqslant \nu_0^X$, then the order becomes (XA, X, SA, S).

Algorithms 3–6 describe the functions used to stimulate a disruption of each type. In these algorithms, we say that a generating decision g^Y, where $Y \in \{S, X, SA, XA\}$, is a candidate if making the decision g^Y produces a schedule of work that respects (i) employee availability/qualifications and (ii) the minimum rest duration between two consecutive shifts. In function `stimulateDisruptionS` (resp. `stimulateDisruptionX`), that is Algorithm 3 (resp. 4), we look for a generating decision g^S (resp. g^X) that swaps (resp. modifies) two shifts assigned to two employees, e_1 and e_2, whose difference in remuneration levels is maximum and for which the exchange of working hours would reduce the time worked by the employee at the largest remuneration level. In this case, the cost of the decision

generated could be significantly negative. Note that, in these two algorithms, the distance between each pair of employees is used in Step 6 to speed up the search by eliminating all pairs with a distance above a predefined threshold $dist^0$. Indeed, when the distance is important between two employees, the flexibility to exchange working hours between the schedules of these two employees is minimal and it becomes difficult (if not impossible) to generate an elementary decision from this generating decision.

Algorithm 3 : Function `stimulateDisruptionS`

input : subproblem $\bar{\mathcal{C}} = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
output : generating decision g^S;

$\bar{k} \leftarrow |\mathcal{K}^L|$;
while $\bar{k} \geqslant 2$ **do**
  foreach $e_1 \in \bar{\mathcal{E}} \cap \mathcal{E}_x(\bar{k})$ **do**
    foreach $k \in \{1, \ldots, \bar{k}-1\}$ **do**
      foreach $h \in \mathcal{H}$ **do**
        foreach $e_2 \in (\bar{\mathcal{E}} \cap \mathcal{E}_x(k)) \backslash \{e_1\} \mid dist_h(e_1, e_2) \leqslant dist^0$ **do**
          $s_1 \leftarrow \mathcal{S}_x(e_1, h)$;
          $s_2 \leftarrow \mathcal{S}_x(e_2, h)$;
          if $l_{s_1} > l_{s_2}$ *and* $g^S(h, e_1, e_2)$ *is a candidate* **then**
            return $g^S(h, e_1, e_2)$;
  $\bar{k} \leftarrow \bar{k} - 1$;
return NIL;

Algorithm 4 : Function `stimulateDisruptionX`

input : subproblem $\bar{\mathcal{C}} = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
output : generating decision g^X;

$\bar{k} \leftarrow |\mathcal{K}^L|$;
while $\bar{k} \geqslant 1$ **do**
  foreach $e_1 \in \bar{\mathcal{E}} \cap \mathcal{E}_x(\bar{k})$ **do**
    foreach $k \in \{1, \ldots, \bar{k}-1\}$ **do**
      foreach $h \in \mathcal{H}$ **do**
        foreach $e_2 \in \bar{\mathcal{E}} \cap \mathcal{E}_x(k) \mid dist_h(e_1, e_2) \leqslant dist^0$ **do**
          $s_1 \leftarrow \mathcal{S}_x(e_1, h)$;
          $s_2 \leftarrow \mathcal{S}_x(e_2, h)$;
          for $l = l_{s_2} - 1$ *to* 1 **do**
            if $g^X(h, e_1, e_2, l)$ *is a candidate* **then**
              return $g^X(h, e_1, e_2, l)$;
  $\bar{k} \leftarrow \bar{k} - 1$;
return NIL;

Algorithm 5 : Function `stimulateDisruptionSA`

input : subproblem $\bar{\mathcal{C}} = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
output : generating decision g^{SA};

foreach $s \in \bar{\mathcal{S}}^A(x)$ **do**
  foreach $k \in \mathcal{K}^L$ **do**
    foreach $e \in \bar{\mathcal{E}} \cap \mathcal{E}_x(k)$ **do**
      $\bar{s} \leftarrow \mathcal{S}_x(e, h_s)$;
      if $l_{\bar{s}} > l_s$ *and* $g^{SA}(h_s, e, s)$ *is a candidate* **then**
        return $g^{SA}(h_s, e, s)$;
return NIL;

Function `stimulateDisruptionSA` (Algorithm 5) allows us to find a generating decision g^{SA}, making an employee work more with the goal of reducing the total length of the anonymous shifts. When

the employee in question does not work, the selected anonymous shift is eliminated. Note that, in Step 4, the levels in $\mathcal{K}^L$ are explored in increasing order to favor increasing the working time of an employee who works less. Finally, in function `stimulateDisruptionXA` (Algorithm 6), the same strategy is used when looking for a generating decision g^{XA} that will eliminate or shorten an anonymous shift $s \in \bar{\mathcal{S}}^A(x)$. Elimination occurs when $l = l_s$ in Step 4.

Algorithm 6 : Function `stimulateDisruptionXA`

input : subproblem $\bar{\mathcal{C}} = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
output : generating decision g^{XA};

1 **foreach** $s \in \bar{\mathcal{S}}^A(x)$ **do**
2 **foreach** $k \in \mathcal{K}^L$ **do**
3 **foreach** $e \in \bar{\mathcal{E}} \cap \mathcal{E}_x(k) \mid \mathcal{S}_x(e, h_s) \neq s_0(h_s)$ **do**
4 **for** $l = l_s$ *to* 1 **do**
5 **if** $g^{XA}(h_s, e, s, l)$ *is a candidate* **then**
6 return $g^{XA}(h_s, e, s, l)$;
7 return NIL;

Observe that applying a generating decision $\widehat{g}^Y$ returned by the Algorithms 3–6 can yield an infeasible solution because not all employee schedule constraints are checked when determining the candidate decisions. This ensures diversity in the search. Note also that the cost of the resulting (feasible or infeasible) solution is less than that of the current solution in most cases.

3.4 PSDH

Algorithm 7 describes the main scheme of PSDH which invokes Algorithms 8–11 that will be described later in this section. PSDH takes as inputs an initial solution x_0 and a parameter ζ, which represents the initial number of subproblems to be created by each of the clustering Algorithms 1 and 2. It is made up of a *while* loop (Steps 2–17) which stops in Step 12 when the solution process is stalling as discussed below. This loop contains an intensification phase (Steps 4–10) and a diversification phase (Steps 14–17). In the intensification phase, an iterative improvement heuristic is executed, invoking at each iteration the function `parallelResolution` (Algorithm 8) to perform a parallel search. For each call, we calculate $t_2 - t_1$, the total time taken by the function to find policy $\delta \in \bar{\Pi}(\bar{x})$, where $\bar{x} \in \mathcal{X}$ is the current solution. If $\varphi^c(\delta) \leqslant c^0(t_2 - t_1)$ where $c^0 < 0$ is a parameter indicating an expected minimum cost reduction by time unit of computation, we estimate that the improvement obtained at this iteration is sufficiently large to pursue the search from the updated solution $\bar{x} + \delta$. Otherwise, the intensification phase stops because this solution is near optimal, or its decomposition by the clustering algorithms has little chance to yield subproblems for which we can create potentially good disruption stimulations. Note that the number of subproblems $\bar{\zeta}$ decreases in Step 10 whenever the number of active subproblems a (i.e., the number of subproblems that return a policy) is less than 25% of the total number of subproblems considered (given by $2\bar{\zeta}$). In this way, we increase the size of the subproblems and, thus, the chance of generating policies that can improve the current solution.

The diversification phase consists in drastically modifying the current solution $\bar{x}$ by calling the function `modifySolution` (Algorithm 11). This function simply removes the schedules of n^P employees, where n^P is set in Step 15 as the maximum number of employees on the same remuneration level. It returns an update current solution $\bar{x}$ and the cost increase c^M incurred by this modification. Following this modification, the number of subproblems $\bar{\zeta}$ to be created by each clustering algorithm is reset to its initial value ζ (Step 17).

This iterative process, alternating between intensification and diversification, is repeated until reaching an iteration where the correction made in the intensification phase does not yield a sufficient cost reduction c^- compared to the cost increase c^M incurred in the last diversification phase (Step 12), i.e., until $c^- < (1+\epsilon)c^M$, where ϵ is a relatively small positive parameter value. Note that this stopping criterion ensures that the algorithm terminates as soon as a cycle occurs (i.e., solution $\bar{x}$ is the same as in the last iteration in Step 12).

Algorithm 7 : PSDH

input : initial solution x_0;
number of subproblems ζ;
output : final solution x^*;

1 $\bar{\zeta} \leftarrow \zeta$, $\bar{x} \leftarrow x_0$, $x^* \leftarrow x_0$, $c^M \leftarrow \infty$;
2 **while** *true* **do**
3 $c^- \leftarrow 0$, $\delta \leftarrow ()$;
4 **while** $\varphi^c(\delta) \leqslant c^0(t_2 - t_1)$ **do**
5 $t_1 \leftarrow$`clock()`;
6 $(\delta, a) \leftarrow$`parallelResolution`$(\bar{x}, \bar{\zeta})$;
7 $t_2 \leftarrow$`clock()`;
8 $\bar{x} \leftarrow \bar{x} + \delta$;
9 $c^- \leftarrow c^- - \varphi^c(\delta)$;
10 **if** $a \leqslant \frac{\bar{\zeta}}{2}$ *and* $\bar{\zeta} \geqslant 2$ **then** $\bar{\zeta} \leftarrow \bar{\zeta} - 1$;
11 **if** $c^\top \bar{x} < c^\top x^*$ **then** $x^* \leftarrow \bar{x}$;
12 **if** $c^M \neq \infty$ *and* $c^- < (1+\epsilon)c^M$ **then** return x^*;
13 **else**
14 $\bar{k} \leftarrow \arg\max_{k \in \mathcal{K}^L} |\mathcal{E}_x(k)|$;
15 $n^P \leftarrow |\mathcal{E}_x(\bar{k})|$;
16 $(\bar{x}, c^M) \leftarrow$`modifySolution`$(\bar{x}, n^P)$;
17 $\bar{\zeta} \leftarrow \zeta$;

Described in Algorithm 8, function `parallelResolution` is the cornerstone of PSDH. It looks for an improving policy $\delta \in \bar{\Pi}(x)$ of the current solution $x \in \mathcal{X}$. This policy is a combination of several compatible policies, found in the 2ζ subproblems taken into consideration. It starts in Step 1 by calling the algorithms `clusteringE` and `clusteringA` that creates ζ subproblems each, which are denoted $(\mathcal{C}_1(x), \mathcal{C}_2(x), \ldots, \mathcal{C}_{2\zeta}(x))$. Then, in Step 3, 2ζ threads are reserved, one for each subproblem.

On each thread $i \in \{1, \ldots, 2\zeta\}$, Steps 5–13 are executed in parallel. On thread i, we attempt to find a policy $\delta_i \in \bar{\Pi}(x, \Phi^n, \mathcal{C}_i(x))$. This policy is formed iteratively by first updating the residual limit $\bar{\varphi}^n$ on the number of modifications allowed in Step 8 before calling function `findElementaryDecision`$(\mathcal{C}_i(x), \bar{\varphi}^n)$ (Algorithm 9) in Step 9. If δ_i contains at least one elementary decision, we say that the subproblem $\mathcal{C}_i(x)$ is active and set $a_i = 1$.

Once the resolution of the subproblems is completed, the following model (P) is solved in Step 14 to find an optimal combination of compatible policies in $\{\delta_i\}_{i \in \{1,\ldots,2\zeta\}}$. In this model, w_i, $i \in \{1, \ldots, 2\zeta\}$, is a binary variable equal to 1 if policy δ_i is selected and to 0 otherwise. Also, define $F = \{(i,j) \in \{1, \ldots, 2\zeta\}^2 \mid i < j, \mathcal{E}_{\delta_i} \cap \mathcal{E}_{\delta_j} \neq \varnothing \vee \mathcal{S}^A_{\delta_i} \cap \mathcal{S}^A_{\delta_j} \neq \varnothing\}$ as the set of pairs of incompatible policies.

$$\text{(P):} \quad \min_{w} \quad \sum_{i=1}^{2\zeta} w_i \varphi^c(\delta_i) \tag{16}$$

$$\text{subject to:} \quad w_i + w_j \leqslant 1, \qquad \forall (i,j) \in F \tag{17}$$

$$w_i \in \{0,1\}, \qquad \forall i \in \{1, \ldots, 2\zeta\}. \tag{18}$$

Constraints (17) ensure that only pairwise compatible policies are selected, while the objective function guarantees the selection of only improving policies. Therefore, the policy adopted in Step 15 belongs to $\bar{\Pi}(x)$.

Next, we describe the function `findElementaryDecision` (Algorithm 9) invoked in Step 9 of Algorithm 8 and which aims at finding a good elementary decision from the current solution x in the subproblem $\bar{C}(x)$ with a number of modifications that does not exceed $\bar{\varphi}^n$. It starts in Step 1 by finding the search order (Y_1, Y_2, Y_3, Y_4) of the disruption types as discussed in Section 3.3, where $Y_i \in \{S, X, SA, XA\}$ and $Y_i \neq Y_j$ if $i \neq j$. In the search order (Y_1, Y_2, Y_3, Y_4), we first stimulate in Step 4 a disruption $\widehat{g}$ of type Y_i using the corresponding function `stimulateDisruptionY`$_i$ (see Algorithms 3–6) before seeking an elementary decision $d^- \in \mathcal{D}(x)$, restricted to the subproblem $\bar{\mathcal{C}}(x)$,

using the function `correctDisruption` (Step 6). If d^- exists, this decision is returned. Otherwise, we repeat with the next disruption type.

Algorithm 8 : Function `parallelResolution`

input : current solution x;
number of subproblems ζ;
output : policy δ;
number of active subproblems a;

1 $\mathcal{C}^E \leftarrow$`clusteringE`(x, ζ), $\mathcal{C}^A \leftarrow$`clusteringA`(x, ζ);
2 $(\mathcal{C}_1(x), \mathcal{C}_2(x), \ldots, \mathcal{C}_{2\zeta}(x)) \leftarrow \mathcal{C}^E \cup \mathcal{C}^A$;
3 `reserveThreads`(2ζ);
4 **foreach** *thread* i **do**
5 $\quad \delta_i \leftarrow ()$, $a_i \leftarrow 0$, $flag \leftarrow true$;
6 $\quad$ **while** $\phi^n(\delta_i) < \Phi^n$ *and* $flag = true$ **do**
7 $\quad\quad flag \leftarrow false$;
8 $\quad\quad \bar{\varphi}^n \leftarrow \Phi^n - \phi^n(\delta_i)$;
9 $\quad\quad d^- \leftarrow$`findElementaryDecision`$(\mathcal{C}_i(x), \bar{\varphi}^n)$;
10 $\quad\quad$ **if** $d^- \neq NIL$ **then**
11 $\quad\quad\quad \delta_i \leftarrow \delta_i \oplus (d^-)$;
12 $\quad\quad\quad flag \leftarrow true$;
13 $\quad$ **if** $\delta_i \neq ()$ **then** $a_i \leftarrow 1$;
14 $w \leftarrow$ `solveModel(P)`;
15 $\delta \leftarrow \bigoplus_{i=1}^{2\zeta} w_i \delta_i$;
16 $a \leftarrow \sum_{i=1}^{2\zeta} a_i$;
17 return (δ, a);

Algorithm 9 : Function `findElementaryDecision`

input : subproblem $\bar{\mathcal{C}}(x) = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
maximum number of modifications $\bar{\varphi}^n$;
output : elementary decision d^-;

1 $(Y_1, Y_2, Y_3, Y_4) \leftarrow$ `order`$\{SA, XA, S, X\}$;
2 $i \leftarrow 1$;
3 **for** $i \in \{1, \ldots, 4\}$ **do**
4 $\quad \widehat{g} \leftarrow$ `stimulateDisruptionY`$_i(\bar{\mathcal{C}})$;
5 $\quad$ **if** $\widehat{g} \neq NIL$ **then**
6 $\quad\quad d^- \leftarrow$ `correctDisruption`$(\bar{\mathcal{C}}, \widehat{g}, \bar{\varphi}^n)$;
7 $\quad$ **if** d^- *exists* **then**
8 $\quad\quad$ return d^-;
9 return NIL;

Function `correctDisruption` performs a truncated enumeration of the possible elementary decisions starting with the generating decision $\widehat{g}_0$. It is inspired by the heuristic of Hassani et al. [15] and presented in Algorithm 10. The elementary decisions generated during the execution of this function are stored in the set D. The set S contains the generating decision sequences currently under study. When one of these sequences does not yield a feasible solution (i.e., it is not an elementary decision), it is extended in different ways and the resulting sequences are added to the set S^+. This function starts with a single sequence, composed of the generating decision $\widehat{g}_0$, in the set S (Step 1). It then goes into a *repeat* loop that checks for each sequence of σ in S if each employee's schedule is feasible (Step 5). This verification is performed by the function `employeesWithInfeasibleSchedule`, which returns all employees E with an infeasible schedule. If this set is empty and the number of modifications associated with the sequence σ does not exceed the maximum allowed $\bar{\varphi}^n$ (Step 6), then this sequence is added to D. Otherwise, the function `lengthenSequence` is called in Step 9 to create new sequences, which are obtained each by adding another generating decision to σ. Such a generating decision must correct the schedule of at least one employee of E. The repeat loop stops when no sequences need to be extended or when the maximum number of generating decisions reaches a predefined limit n^g.

In the end, the function `correctDisruption` returns the elementary decision, d^-, with the least cost (Steps 12–13).

Algorithm 10 : Function `correctDisruption`

input : subproblem $\bar{\mathcal{C}}(x) = (\bar{\mathcal{E}}, \bar{\mathcal{S}}^A(x))$;
maximum number of modifications $\bar{\varphi}^n$;
maximum number of generating decisions n^g;
generating decision $\hat{g}_0$;
output : elementary decision d^-;

1 $D \leftarrow \varnothing$, $\kappa \leftarrow 1$, $S \leftarrow \{(\hat{g}_0)\}$;
2 **repeat**
3 $S^+ \leftarrow \varnothing$;
4 **for** $\sigma \in S$ **do**
5 $E \leftarrow$ `employeesWithInfeasibleSchedule`$(\bar{\mathcal{E}}, \sigma)$;
6 **if** $E = \varnothing$ *and* $\varphi^n(\sigma) \leqslant \bar{\varphi}^n$ **then**
7 $D \leftarrow D \cup \{\sigma\}$;
8 **else**
9 $S^+ \leftarrow S^+ \cup$ `lengthenSequence`$(\bar{\mathcal{C}}, \sigma, E)$;
10 $S \leftarrow S^+$, $\kappa \leftarrow \kappa + 1$;
11 **until** $S = \varnothing$ *or* $\kappa = n^g$;
12 $d^- \leftarrow \arg\min_{d \in D} c_d$;
13 return d^-;

Finally, the function `modifySolution` (Algorithm 11), which is called in Step 16 of Algorithm 7 to diversify the search, modifies the current solution x by changing the schedule of n^P employees. We choose the employees who work the most in solution x. We transform, thereafter, all shifts assigned to those employees to anonymous shifts (Steps 5–6). This yields a new solution $\bar{x}$ with at least n^P non-working employees and new anonymous shifts. In the next intensification phase, PSDH will seek to assign these new anonymous shifts to employees and balance the total time worked by employees. We denote by c^M the added cost due to these modifications.

Algorithm 11 : Function `modifySolution`

input : feasible solution x;
number of schedules to modify n^P;
output : modified feasible solution $\bar{x}$;
cost of modification c^M;

1 $k \leftarrow |\mathcal{K}^L|$, $i \leftarrow 0$, $\bar{x} \leftarrow x$;
2 **foreach** $e \in \mathcal{E} \cap \mathcal{E}_x(k)$ **do**
3 $i \leftarrow i + 1$;
4 **foreach** $h \in \mathcal{H} \mid \mathcal{S}_x(e,h) \neq s_0(h)$ **do**
5 $\mathcal{S}^A(\bar{x}) \leftarrow \mathcal{S}^A(\bar{x}) \cup \{\mathcal{S}_x(e,h)\}$;
6 $\mathcal{S}_{\bar{x}}(e,h) \leftarrow s_0(h)$;
7 **if** $i = |\mathcal{E} \cap \mathcal{E}_x(k)|$ **then** $k \leftarrow k - 1$;
8 **if** $i = n^P$ **then**
9 $c^M \leftarrow c^\top \bar{x} - c^\top x$;
10 return $(\bar{x}, c^M)$;

4 Numerical results

To test the efficiency of PSDH, we carried out experimental tests on real instances. For these tests, we compared PSDH to three other algorithms:

- an exact algorithm E that we use as a reference for the quality of the solutions returned by PSDH and that solves model (1)–(11) using the parallel mixed-integer programming CPLEX solver (version 12.9.0.0);

- two heuristics, H^1 and H^2, which represent commonly used heuristic variants of the exact algorithm obtained by restricting the set of shifts considered (see below) and by applying a tolerance on the optimality gap of 0.01%.

All algorithms were implemented in C++. All tests were run on a Linux machine equipped with a 12-core (2 threads per core) Intel Core i7 processor clocked at 3.4 GHz with 16 GB RAM. The OpenMP interface was used for parallel computing on shared memory architecture.

In Section 4.1, we describe the instances used, the subsets of shifts considered in heuristics H^1 and H^2, and the parameter setting of PSDH. In Section 4.2, we analyze the results obtained.

4.1 Instances, shift subsets and PSDH parameter setting

Our industrial partner provided us with 14 real instances for which the number of employees varies between 17 and 94 and the number of jobs varies between 2 and 10. Over a one-week horizon divided in 15-minute periods, we have a demand, expressed by a number of employees, for each period and each job in $\mathcal{W}$. For each employee $e \in \mathcal{E}$, we have his qualifications $\mathcal{W}_e$, his availability $\mathcal{P}_e$, the minimum duration $\underline{l}_e$ and maximum duration $\bar{l}_e$ of authorized work per day, and the minimum rest duration r_e required between two consecutive shifts. In addition, we have the minimum length $\underline{l}^A$ and maximum length $\bar{l}^A$ for anonymous shifts, namely, 16 and 40 periods (4 and 10 hours), respectively. The minimum number of days off per week and minimum number of periods of rest between two consecutive shifts are set as follows: $n^O = 2$ days and $n^R = 48$ periods (12 hours).

From these instances, we generated new instances[1] by modifying the qualifications or availability of employees, keeping only instances that have a structure that is sufficiently different from that of their corresponding initial instance. In total, there are 33 test instances. The instances are denoted I_m^n, where $m \in \{1, \ldots, 14\}$ is the number of the original instance and n is a number attributed to the specific instance derived from original instance m ($n = 1$ is the original instance and $n > 1$ indicates a modified instance). For example, I_4^1 is the fourth original instance and I_4^3 is the second instance ($n - 1 = 2$) constructed from the fourth original instance. Some characteristics of these instances are shown in Table 1, namely, the number of employees (*# emp*), the number of jobs (*# jobs*), the average and maximum number of qualified jobs per employee (*# jobs/emp*), the maximum demand per job and period (*max dem*) and the total numbers of personalized and anonymous shifts in the sets $\mathcal{S}_h^e$ and $\mathcal{S}_h^A$ (*# shifts*).

To generate for each day $h \in \mathcal{H}$ the set of potential shifts $\mathcal{S}_h^e$ for each employee $e \in \mathcal{E}$ and the set of anonymous shifts $\mathcal{S}_h^A$, we define p^+, the increment step of the start time of the shifts. Then, beginning with the first available period of employee e on day h, if any, we add in $\mathcal{S}_h^e$ the shifts of all valid durations but only at every p^+ starting periods. The same rule is applied to build set $\mathcal{S}_h^A$. It is easy to see that when $p^+ = 1$, the generation of shifts is exhaustive (algorithm E). For heuristics H^1 and H^2, we used an increment step $p^+ = 2$ and $p^+ = 3$, respectively.

Table 2 presents the PSDH parameter values used for our tests. These values were selected based on preliminary computational results.

4.2 Results and analysis

We start by presenting the results obtained by the three algorithms H^1, H^2 and E in Table 3. In fact, we consider the results of three variants of algorithm E that were executed with a tolerance on the optimality gap of 5%, 1%, and 0.01%. For each instance, Table 3 indicates the total wall-clocked computational times (T) in seconds required for all algorithms. Furthermore, using the solution returned by algorithm E as a reference, it also reports the relative error (Err) made by the other algorithms (as well as PSDH below). The last row in this table provides averages over all instances.

[1] In agreement with our industrial partner Kronos, the instances will be published on a public website if the paper is accepted for publication.

Table 1: Characteristics of the instances

Instance			*# jobs/emp*		*max*	*# shifts*	
	# emp	*# jobs*	*avg*	*max*	*dem*	*personalized*	*anonymous*
I_1^1	17	2	1.2	2	5	278894	24150
I_1^2	17	2	1.2	2	5	418796	24150
I_1^3	17	2	2	2	5	427028	24150
I_2^1	27	2	1.1	2	7	193635	24150
I_2^2	27	2	2	2	7	531272	24150
I_2^3	27	2	1.1	2	7	166494	24150
I_3^1	34	2	2	2	10	904816	48300
I_3^2	34	2	2	2	10	1675184	48300
I_3^3	34	2	2	2	10	606824	48300
I_4^1	54	2	2	2	14	332988	24150
I_4^2	54	2	1.2	2	14	219412	24150
I_4^3	54	2	2	2	14	1062544	24150
I_5^1	34	4	2.4	4	5	901068	48300
I_5^2	34	4	4	4	5	1675184	48300
I_5^3	34	4	2.4	4	5	891356	48300
I_6^1	47	4	2.3	4	7	503636	48300
I_6^2	47	4	4	4	7	1222095	48300
I_6^3	47	4	2.3	4	7	649700	48300
I_7^1	54	4	2.6	4	7	774540	48300
I_7^2	54	4	2.6	4	7	762806	48300
I_8^1	94	4	4	4	14	1056634	48300
I_9^1	25	5	5	5	10	2727718	60375
I_9^2	25	5	2.9	5	10	1625464	60375
I_{10}^1	50	5	5	5	20	4445674	60375
I_{10}^2	50	5	5	5	20	5455436	60375
I_{10}^3	50	5	2.1	4	20	1294363	60375
I_{11}^1	37	7	4.8	7	6	2523292	84525
I_{11}^2	37	7	7	7	6	4300065	84525
I_{11}^3	37	7	4.8	7	6	1136583	84525
I_{12}^1	72	7	7	7	12	5467202	84525
I_{13}^1	94	8	8	8	7	2014544	96600
I_{14}^1	50	10	10	10	10	10910872	120750
I_{14}^2	50	10	10	10	10	4343500	120750

Table 2: PSDH parameter values

Parameter	*Value*
ν_0^Y	0.1 for all $Y \in \{S, X, SA, XA\}$
$dist^0$	0.5
c^0	-20
ϵ	0.05
n^g	2
ζ	4 if $\lvert\mathcal{E}\rvert < 30$ 6 if $30 \leqslant \lvert\mathcal{E}\rvert < 50$ 10 if $50 \leqslant \lvert\mathcal{E}\rvert < 70$ 12 if $70 \leqslant \lvert\mathcal{E}\rvert < 100$

First, we notice that obtaining a near-optimal solution (i.e., within an optimality gap of 0.01%) is very time-consuming, as expected, because algorithm E takes into consideration all feasible shifts for each employee, as well as all possible anonymous shifts ($p^+ = 1$). Note that the algorithm E variants that consider an optimality gap of 1% and 5% yield solutions with an average relative error that is much less than the tolerance (0.36% and 1.87%, respectively), but the computational times remain relatively important (with averages of 1881 and 1257 seconds, respectively).

For Algorithm H^1, we consider a smaller number of shifts, which reduces the computational times. However, this reduction is not significant enough in most instances. We also obtain errors greater than 5% for 24.2% of the instances. With Algorithm H^2, the errors are between 6.2% and 17.9%, whereas the computational times range from 10 minutes to 251 minutes for 30.3% of the instances. It can be concluded that increasing the increment step p^+ during a priori shift generation reduces the

Table 3: Results of algorithms E, H^1 and H^2

	Algo H^1		*Algo* H^2		*Algo* E *(5%)*		*Algo* E *(1%)*		*Algo* E
Instance	*Err (%)*	*T (s)*	*Err (%)*	*T (s)*	*Err (%)*	*T (s)*	*Err (%)*	*T (s)*	*T (s)*
I_1^1	3.3	42	9.0	27	0.2	42	0.2	119	146
I_1^2	2.2	79	7.8	23	0.1	101	0.1	98	125
I_1^3	2.0	50	8.2	49	0	147	0	151	149
I_2^1	4.2	160	11.1	102	0.2	175	0.1	186	981
I_2^2	4.9	180	9.8	216	0.2	490	0.2	489	1737
I_2^3	4.2	267	10.0	172	1.2	403	0.9	398	1112
I_3^1	3.3	71	9.7	27	0.1	119	0	119	168
I_3^2	3.4	40	9.3	26	0.1	103	0.1	102	152
I_3^3	3.3	37	9.1	27	4.8	64	0	88	100
I_4^1	4.2	1542	11.2	890	2.3	428	0.2	435	7674
I_4^2	3.9	1814	10.8	226	0.4	407	0.4	406	6454
I_4^3	4.2	1454	9.8	2223	0.9	781	0.9	778	9158
I_5^1	3.3	247	9.7	65	3.2	287	0.1	306	455
I_5^2	4.6	613	9.0	112	3.3	820	1	819	1712
I_5^3	3.3	144	8.8	1068	0.2	913	0.2	918	933
I_6^1	2.8	648	7.7	407	0.9	228	0.9	230	5375
I_6^2	2.9	1914	6.6	385	4.7	1641	1	2123	10119
I_6^3	2.4	10037	6.2	855	4.9	3157	0	10073	10084
I_7^1	4.1	5031	11.0	4663	1.7	1443	0.7	1516	10081
I_7^2	4.2	2816	10.6	10033	2.6	1390	0.7	1409	9882
I_8^1	2.7	18589	7.7	574	4.3	2992	0.5	13608	20100
I_9^1	5.2	286	13.9	166	0.3	596	0.3	572	601
I_9^2	5.1	168	13.6	147	0	246	0	251	346
I_{10}^1	5.4	459	11.3	303	0.6	1017	0.6	1013	1025
I_{10}^2	4.0	451	13.4	286	0	1095	0	1084	1105
I_{10}^3	4.7	293	13.6	115	1.2	574	0.9	683	851
I_{11}^1	6.9	425	17.2	213	4.3	2054	0.7	2027	2621
I_{11}^2	6.0	382	15.4	382	5	766	0.3	787	896
I_{11}^3	4.3	978	17.9	208	1.8	978	0.2	830	1068
I_{12}^1	6.8	19696	6.8	15113	3.5	4192	0	6762	18874
I_{13}^1	2.7	20230	7.6	1730	0.5	5100	0.5	5104	20451
I_{14}^1	5.2	1875	13.9	1372	3.5	4381	0	4299	5840
I_{14}^2	5.8	1807	11.2	1262	4.9	4339	0.3	4286	4266
Average	4.11	2813	10.60	1317	1.87	1257	0.36	1881	4686

computational time but at the expense of significantly deteriorating solution quality. Furthermore, our results show that it seems more efficient to use algorithm E with a prefixed tolerance on the optimality gap than to use an approximate model with a reduced set of potential shifts (algorithms H^1 and H^2).

PSDH, in contrast, is a dynamic algorithm that is not based on the initial generation of a potential set of shifts. Indeed, it modifies the shifts present in the current solution during the resolution, using the generating decisions g^S, g^X, g^{SA} and g^{XA}, in order to create new ones. On the other hand, PSDH requires an initial solution x_0. To compute this solution, we applied another heuristic variant of algorithm E where the objective function aims only at minimizing the over-coverage penalties. The resulting schedule is characterized by the significant presence of anonymous shifts, as well as a significant difference in the total time worked by each employee (the value $var(P_{x_0}(\mathcal{E}))$ is close to 1). Even though the solution x_0 has a very significant error (between 45.3% and 106% for the tested instances), it represents a very rich field to find elementary decisions with a negative cost.

Figure 1 (where the error is given as a function of time for two selected instances) demonstrates how the error decreases rapidly in the first phase of the solution process. When PSDH starts to take longer to improve on the current solution (when the triangles in Figure 1 become more spaced apart), it calls the function `modifySolution`, resulting in an error peak (due to the added modification cost c^M). In the same figure, we observe how PSDH can in some cases quickly shifts to an improved solution compared to the one found before the function `modifySolution` is called.

Table 4 discloses the results obtained by PSDH on all test instances. Its columns are divided in three parts. Columns 2 and 3 indicate the error of the initial solution and the computational time

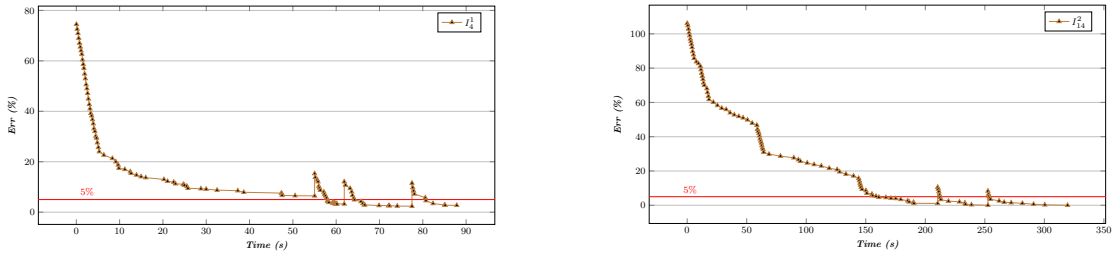

Figure 1: Reduced error in terms of time for instances I_4^1 and I_{14}^2

required to compute it. As mentioned above, the average quality of these solutions is poor (with an average error of 65%) but they are computed in an average time of 59 seconds, which is negligible compared to the average times obtained in Table 3.

Table 4: PSDH results

Instance	*Initial solution*		*PSDH*		*PSDH-600s*	
	Err (%)	*T (s)*	*Err (%)*	*T (s)*	*Err (%)*	*Gain (%)*
I_1^1	63.4	12	1.1	20	1.1	0
I_1^2	63.8	12	0.5	5	0.3	34
I_1^3	53.8	12	0.9	29	0.9	0
I_2^1	71.1	9	2.2	49	1.4	36
I_2^2	59.7	14	0.2	27	0.2	12
I_2^3	71.1	9	2.4	54	1.8	25
I_3^1	76.2	27	4.7	63	4.3	7
I_3^2	81.1	12	5.5	103	2.0	64
I_3^3	79.2	12	4.5	61	1.7	63
I_4^1	74.5	10	2.4	88	1.8	25
I_4^2	74.5	7	3.1	98	2.4	23
I_4^3	62.4	18	0.7	63	0.7	0
I_5^1	73.2	30	4.4	39	4.4	0
I_5^2	58.4	41	1.7	19	1.3	25
I_5^3	74.7	28	0.6	39	0.6	0
I_6^1	53.7	22	4.0	164	4.0	0
I_6^2	38.1	36	0.9	70	0.9	2
I_6^3	51.4	33	0.9	74	0.4	56
I_7^1	65.8	24	4.3	56	2.0	54
I_7^2	65.7	24	1.9	114	2.0	0
I_8^1	51.2	28	2.6	309	2.6	0
I_9^1	56.9	65	1.3	61	0.2	84
I_9^2	55.9	25	2.7	30	2.7	0
I_{10}^1	59.2	91	3.9	345	1.6	58
I_{10}^2	58.9	93	1.1	212	1.1	0
I_{10}^3	59.4	35	2.3	212	1.8	22
I_{11}^1	62.7	82	0.2	53	0.1	76
I_{11}^2	66.7	83	0.04	92	0.04	0
I_{11}^3	62.4	84	0.4	44	0.4	0
I_{12}^1	71.6	132	1.4	117	1.2	12
I_{13}^1	45.3	118	3.2	273	1.0	68
I_{14}^1	105.0	356	0.4	541	0.4	0
I_{14}^2	106.1	355	0.0	319	0	0
Average	65.85	59	2.01	116	1.43	22.6

The next two columns provide the same statistics for PSDH (here, Time excludes the time to compute the initial solution). We can see that for each instance, except for instance I_3^2, we obtain an acceptable error (less than 5%), in an average time of 175 seconds (including the time to compute the initial solution). For 82% of the instances, this time is less than 5 minutes. Consequently, PSDH succeeds to compute good-quality solutions (with an average error of 2.01%) much faster than the variants of algorithm E. The difference in the computational times is more striking for the largest instances, namely, the 13 instances with 50 employees or more: the average time for PSDH (including the time to compute x_0) is 311 seconds, while algorithm E with a tolerance of 5% requires 2165

seconds on average to produce solutions with approximately the same average error (see Table 3). For these instances, PSDH yields an average error of 2.10%, which is very acceptable given the usual precision of the demand forecast (the average error is 2.03% for the solutions produced by algorithm E with a 5% tolerance). Note that, according to our industrial partner, fast computational times are highly valuable for large retail companies with a very large number of branches that compute all their employee schedules at a central location or using a cloud computing service which they pay according to the computational time used.

Finally, the last two columns report the results of a slightly modified version of PSDH, denoted PSDH-600s. In fact, the only difference concerns the stopping criterion in Step 12 of Algorithm 7 which is replaced by a time limit of 600 seconds (excluding the time to compute the initial solution). These results permits to assess the ability of PSDH to compute a better solution when additional time is allowed. The first of these two columns specify the error obtained and the second the percentage of gain on the error with respect to the original version of PSDH. As it can be observed, allowing more time can reduce the error for 19 of the 33 test instances, with an average reduction percentage of 22.6%. This percentage is very significant for some instances (reaching 84% for instance I_9^1). For the other instances where this percentage is low or zero, except I_3^1 and I_5^1, we notice that the solution found by PSDH is already of very good quality, and PSDH-600s is unable to surpass it in quality.

5 Conclusion

In this paper we considered the problem of optimizing a personnel schedule in a context where employees can be assigned to a wide variety of shifts, starting and ending at various times. We proposed a fast heuristic, PSDH, based on the stimulation/correction of certain disruptions. The choice of the disruption type is based on scores that approximate the probability that correcting such a disruption will yield a better solution. The use of two clustering algorithms, as well as the notion of compatibility between policies, allowed us to use parallel computing. Computational results obtained on 33 instances (14 of which are real) have shown the effectiveness of PSDH. For 97% of the tested instances, PSDH found solutions that can be deemed of very good quality (error less than 5%) and, for 82% of the instances, these solutions were computed in less than 5 minutes. The tests also proved the effectiveness of the dynamic generation of the potential shifts.

We believe that PSDH can be applied in other contexts. Doing so will require two adaptations: (i) adapting the generating decisions to the new context, and (ii) adapting the scores ν^Y. We also envision that a solution found by PSDH will be easily re-optimizable when minor or large disruptions are revealed, or in case of interactive optimization. Indeed, any modification imposed on the solution found can be viewed as a call to the function `modifySolution`. Thus, changes between employee schedules will be flexible during re-optimization, given that the solution to be re-optimized had been found with the same types of modifications.

References

[1] Beaulieu, H., Ferland, J.A., Gendron, B., Michelon, P., 2000. A mathematical programming approach for scheduling physicians in the emergency room. Health Care Management Science 3, 193–200.

[2] Bechtold, S.E., Jacobs, L.W., 1990. Implicit modeling of flexible break assignments in optimal shift scheduling. Management Science 36, 1339–1351.

[3] Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W., 2008. Scheduling Breaks in Shift Plans for Call Centers, in: Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling, Montréal, Canada.

[4] 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, 367–385.

[5] Bonutti, A., Ceschia, S., De Cesco, F., Musliu, N., Schaerf, A., 2017. Modeling and solving a real-life multi-skill shift design problem. Annals of Operations Research 252, 365–382.

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

[7] Dantzig, G.B., 1954. Letter to the editor-A comment on Edie's traffic delays at toll booths. Journal of the Operations Research Society of America 2, 339–341.

[8] Di Gaspero, L., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A., Slany, W., 2007. The minimum shift design problem. Annals of Operations Research 155, 79–105.

[9] Edie, L.C., 1954. Traffic delays at toll booths. Journal of the Operations Research Society of America 2, 107–138.

[10] 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, 21–144.

[11] Felici, G., Gentile, C., 2004. A polyhedral approach for the staff rostering problem. Management Science 50, 381–393.

[12] Gaballa, A., Pearce, W., 1979. Telephone sales manpower planning at Qantas. Interfaces 9, 1–9.

[13] Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

[14] Hansen, P., 1986. The steepest ascent mildest descent heuristic for combinatorial programming, in: Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, pp. 70–145.

[15] Hassani, R., Desaulniers, G., Elhallaoui, I., 2021. Real-time bi-objective personnel re-scheduling in the retail industry. European Journal of Operational Research (Forthcoming).

[16] Jacquet-Lagrèze, E., Montaut, D., Partouche, A., 1998. The shift scheduling problem: Different formulations and solution methods. Foundations of Computing and Decision Sciences 23, 199–217.

[17] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by Simulated Annealing. Science 220, 671–680.

[18] 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.

[19] Miller, H.E., Pierskalla, W.P., Rath, G.J., 1976. Nurse scheduling using mathematical programming. Operations Research 24, 857–870.

[20] Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24, 1097–1100.

[21] Moondra, S.L., 1976. An LP model for work force scheduling for banks. Journal of Bank Research 7, 299–301.

[22] Musliu, N., Schaerf, A., Slany, W., 2004. Local search for shift design. European Journal of Operational Research 153, 51–64.

[23] Rekik, M., Cordeau, J.F., Soumis, F., 2010. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. Journal of Scheduling 13, 49–75.

[24] Warner, D.M., 1976. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. Operations Research 24, 842–856.

[25] Widl, M., Musliu, N., 2014. The break scheduling problem: complexity results and practical algorithms. Memetic Computing 6, 97–112.