# A two-phase method for the Shift Design and Personnel Task Scheduling Problem with Equity objective

Damien Prot, Tanguy Lapègue, Odile Bellenguez-Morineau

# A two-phase method for the Shift Design and Personnel Task Scheduling Problem with Equity objective

Damien Prot, Tanguy Lapègue, Odile Bellenguez-Morineau

L'UNAM Université, École des Mines de Nantes, IRCCyN UMR CNRS 6597
(Institut de Recherche en Communication et en Cybernétique de Nantes),
4 rue Alfred Kastler - La Chantrerie BP20722
44307 NANTES Cedex 3 - FRANCE

**Abstract**

In this paper, we study the Shift Design and Personnel Task Scheduling Problem with Equity objective (SDPTSP-E), initially introduced in [11]. This problem consists in designing the shifts of workers and assigning a set of tasks to qualified workers, so as to maximise the equity between workers. We propose a natural two-phase approach consisting in first designing shifts and then assigning tasks to workers, and we iterate between these two phases to improve solutions. We compare our experimental results with existing literature and show that our approach outperforms previous known results.

## 1 Introduction

In personnel scheduling problems, the aim is to build employee rosters that respect legal and organisational constraints. The aim is also often to propose rosters that fit with employee wishes and/or that are fair between employees in order to maximise the staff satisfaction. In this paper, we study the Shift Design and Personnel Task Scheduling Problem with Equity objective (SDPTSP-E), initially introduced in [11].This problem consists in designing the shifts of workers and assigning a set of tasks to skilled workers, so as to maximise the equity between workers. In [11], an integrated approach is proposed. It simultaneously deals with the assignment of tasks and the design of rosters. In this paper, we split this problem in two steps: first designing shifts and then assigning tasks to workers; we iterate between these two phases in order to improve our solution.

A state of the art for the SDPTSP-E is presented in [11]. More precisely, papers related to Nurse Rostering Problem (see [4] for a survey), Tour Scheduling Problem (see [15] and [1] for a survey), Personnel Task Scheduling Problem (see

[9]) and Fixed Job Scheduling Problem (FJSP, see [7] and [8] for two surveys) are analyzed and it is shown that our problem shares characteristics with all these problems but always differs in such a way that it is not possible to use existing methods. For example, in many personnel scheduling problems (see [5] for a survey up to 2004 and [3] for 2004-2012), the design of the shifts is an input for the problem, whereas we have to define the shifts in our problem. Moreover, the workload is often given as a number of required workers per skill, per time period, whereas it is given by a set of fixed tasks that cannot be preempted in our case. The FJSP, which address this problem does not take into account legal constraints. Moreover, to the best of our knowledge, the objective function (the equity among workers) we are taking into account has never been studied in such problems. The closest objective function can be found in [14] for a crew rostering problem; the authors are considering a weighted sum of three average relative deviations, one of them being the deviation between the ideal and the real monthly flight time raised to the power $r$ ($r = 3$ in their experiments).

When considering employee scheduling problems, two-phase methods are very common. The main idea of these methods is to solve two subproblems, sometimes optimally, although there is no guarantee that the optimal solution of the overall problem can be found. For instance, in [16], the authors propose an algorithm using two steps for the nurse rostering problem, where in the first phase the working days are computed while in the second one the type of shift assigned each day is computed. Likewise, in [6], the authors study the minimum cost shift scheduling problem with a guaranteed service level, and propose a two-step heuristic; in the first phase, staffing requirements are computed whereas in the second one a minimum cost schedule is found. In [13] the authors propose a two-stage procedure for the multi-activity and task assignment problem, consisting of assigning interruptible activities and uninterruptible tasks to given work shifts so as to match as much as possible for each activity a time-dependent demand curve.

## 2 Problem description

The problem we study in this paper is the Shift Design and Personnel Task Scheduling Problem with Equity objective (SDPTSP-E), which has been introduced in [11]. For the sake of completeness, we recall here the main features of the problem. A set of tasks, with fixed starting and finishing times (lying in a time horizon of one week), has to be performed by a set of workers, while trying to minimise the inequity among workers. In this problem, a task corresponds to a fixed interval of work that cannot be preempted and requires exactly one qualified worker. We call *shift* of a worker the time he/she spends at the office, i.e. a shift corresponds to a period when the worker is available to perform tasks. Note that the shifts can be computed in two ways: first, the shift of a worker can be computed a priori, and hence we have to assign to this worker only tasks lying in his/her shifts. Second, some tasks may be first assigned to a worker, and the shifts are computed relatively to the allocated tasks. In both

cases, task assignment and shifts have to respect specific constraints.

## 2.1 Constraints

Two different types of constraints have to be respected while designing shifts and assigning tasks to workers, the organisational and the legal ones.

### 2.1.1 Organisational constraints

The organisational constraints are the following ones:

- Employees cannot perform tasks which require unmastered skills.
- Employees cannot perform tasks while unavailable.
- Every task must be assigned to one employee.
- The assignment of meetings must be respected.
- Employees must finish a task before starting another one.
- Tasks starting after 6 a.m. on a given day must not belong to the same shift as tasks starting before 6 a.m.

This last constraint needs explanations: Tasks starting before 6 a.m. should be considered as night tasks, meaning that they should be performed during a night shift, whereas tasks starting after 6 a.m. should be considered as morning tasks, meaning that they should be performed during a morning shift. The purpose of this constraint is to avoid shifts starting in the middle of the night and ending in the middle of the morning, because they are not appreciated by employees.

### 2.1.2 Legal constraints

In the following, we introduce the notion of daily working time and duration of a working day. Note that there is a difference between these two terms: If the shift starts before noon and ends after 2:30 p.m., then a one-hour lunch break has to be removed from the duration to obtain the working time. For example, if a shift starts at 11 a.m. and ends at 5 p.m., the corresponding working time is equal to 5 hours. We also use the notion of working day for an employee: A day is considered as worked if the employee is working between 6 a.m. and 6 a.m. in the next morning.

Legal constraints are the following ones:

- The daily working time must not exceed 10 hours.
- The weekly working time must not exceed 48 hours.
- The duration of a working day must not exceed 11 hours.
- The duration of a rest period must not be less than 11 hours.

- The duration of the weekly rest must not be less than 35 hours.
- Series of consecutive working days must not exceed 6 days.
- Employees do not work more than one shift per working day.
- Depending on the starting and ending times of their shifts, nurses must have a lunch break or not.

This last constraint needs further explanations. In [11] the authors decided to deal with lunch break, by guaranteeing that an employee who should have a lunch break does not work during at least one hour in its shift. This correspond to the industrial way of dealing with lunch break. Note that the break is not necessarily placed in the time window $[12a.m.; 2:30p.m.]$, since it is possible that an employee has to do a long task that completely overlaps lunch hours. Hence, if it is the case and if we want that the lunch break lies in this interval, we are sure that no feasible solution exists. In practice, when such a case occurs, the employee is flexible and does a break before or after such a task. In order to deal correctly with the lunch break, we hence decide to bound the sum of the processing times of tasks assigned to a worker in a given shift by the maximum between the daily working time and five hours (this corresponds to the length of the longest tasks in our instances). It means that, if a shift is starting at 11 a.m. and ending at 7 p.m., then the sum of tasks' processing times can not exceed seven hours (we have to keep one hour for the lunch break). If the shift is starting at 11 a.m. and ending at 3 p.m., then the sum of processing times is bounded by four hours (no need to keep time for the lunch break, if the length of the shift is less than five hours).

## 2.2 Equity

Legal and organisational constraints must be satisfied, but they only ensure the feasibility of schedules. In order to build good schedules, the workload should be shared among workers in a fair way. Basically, the idea is just to find a solution where employees have the same amount of work. However, in this paper, we only deal with fixed tasks, and thus, we only deal with a part of the global workload. Apart from fixed tasks, employees also have some administrative work to do. This work is not fixed, may be preempted and is already assigned to employees, which explains why it is not taken into account in this study. Moreover, some employees may have more administrative work to do than others, which means that these employees cannot perform as many fixed tasks as other employees. Consequently, it is required to distinguish workers that have a lot of administrative work to perform from the others. That is why the decision-maker associates with each employee a weekly *targeted working load*, corresponding to the time each worker should dedicate to the fixed tasks.

The idea of our objective function is to be as close as possible to this value for each worker. Indeed, if a worker is above his/her targeted working load, then he/she will have great difficulties to perform all his/her administrative work.

On the contrary, a worker that is under his/her targeted working load, will have a lot of free time. Of course, workers prefer to be under their targeted working load, but we assume that the sum of the processing times of the fixed tasks is equal to the sum of the targeted working loads. Therefore, if some workers are under their targeted working load, it means that other workers are above, which is not fair.

The objective function we consider in this paper is the *equity* among workers, that we want to maximise. It is equivalent to minimise the inequity, defined as the difference between the highest and the lowest worker's gap value, itself being the difference $\Delta_i$ between the targeted working load and the real working load of the worker $w_i$. We hence want to minimise $\Delta = \max_i \Delta_i - \min_i \Delta_i$. A small difference between the highest gap and the smallest gap means that the workload is well balanced among workers. This objective criterion is illustrated on Figure 1. In figure 1a, the difference between the targeted and the real working load is null for the worker $w_2$, but not for $w_1$ and $w_3$, leading to a strictly positive value of the objective function. On the contrary, on Figure 1b, the real working load perfectly matches with the targeted working load for each worker, and we hence have an optimal solution.

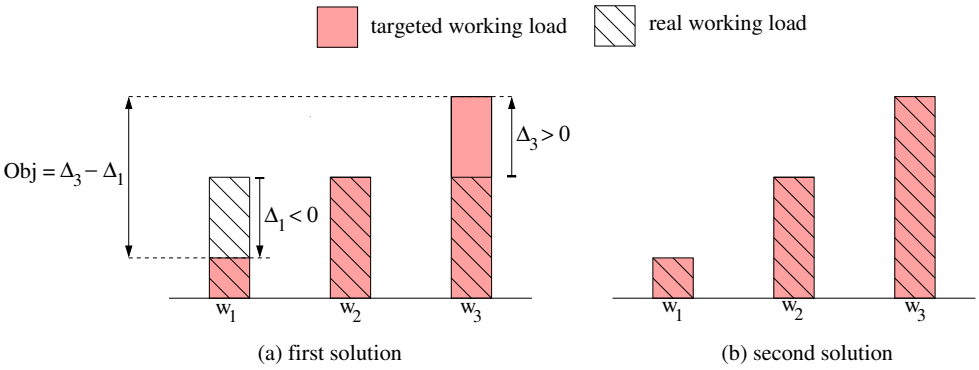


Figure 1: Objective function: equity

## 2.3 Feasibility

Sometimes, it is not possible to find a solution that respects all these constraints. In practice, this may be handled in two different ways. The decision-maker may first try to find agreements with employees in order to design a few longer days, or a few shorter rests. If such agreements cannot be found, or if the workload is simply too difficult to share among the regular workforce, then the decision-maker hires externals to strenghten the regular workforce. It could be the case when there exist some activity peaks where a number of simultaneous workers, bigger than the regular employees, is needed. In this case, the decision-maker tries to hire as few as possible externals, which requires a deep insight on

the problem at hand to evaluate the different time windows where additional workforce is needed on an acceptable timetable for regular workers. Therefore, in this paper (as in [11]), we propose to relax the constraint which specifies that each task has to be assigned to exactly one employee. Our approach, is consequently able to produce solutions with a partial assignment (i.e. some tasks are left unassigned). These solutions are useful to the manager because they provide insights regarding to the need on externals. However, a complete assignment is always better than a partial one. Likewise, given two different partial solutions, the one with the smallest number of unassigned tasks is also always better, because it provides sharper insights on the problem at hand. More formally, in this paper, the minimisation of the number of unassigned tasks and the minimisation of the inequity among workers are handled in a lexicographic order.

## 2.4 General remarks regarding the design of shifts

One may wonder how realistic it is to work on personnel schedules with a precision of one minute. Indeed, in the literature, especially in nurse rostering, it is classical to work with a small set of predefined shifts, and try to have a regular pattern for each worker. It is not possible to work with such a set, because some tasks can last up to five hours, hence leading to a large set of predefined shifts in order to cover the this kind of tasks all over the day. Nevertheless, we try to adapt this two-phase method with predefined shifts (from four to eight 8-hours and/or 10-hours predefined shifts), and the results are not conclusive, since in the best case we are able to find only 60% of the complete solutions we can have with our approach because there still exist uncoverable tasks or activity peaks. Even on the extreme case where we predefine 24 different patterns (one shift starts each hour of the day), the way of dealing with tasks assignment is still not flexible enough (to manage different shift lengthes for example) and leads to uncompleted assignments that enforce to hire more externals than needed, which is unacceptable.

Moreover, in our case, the workload is fixed with a precision of one minute because of medical constraints and it cannot be changed. The decision maker wants to check precisely the feasibility of schedules, and workers are willing to respect the imposed timing. Consequently, the assignment of tasks to workers is handled with a precision of one minute. However, this does not mean that the design of shifts has to be done with a precision of one minute. Indeed, if we consider a shift composed of two tasks of one hour, starting respectively at 7h14 and at 13h42, then the worker assigned to this shift may choose to start working any time before 7h14, and may leave any time after 14h42. The employee may use this extra working time to work on his/her administrative work. However, the employee, may also prefer to have a smaller shift (7h14 to 14h42). This choice is up to the employee, and it is not easy to determine a priori what kind of extension the employee is going to choose, because it depends on many criteria (wishes, density of the working day, kind of tasks assigned). Therefore, the most natural way to deal with shifts is to consider the case without any

extension. Note also that depending on the end (resp. start) of the previous (resp. next) working day, extensions may be limited. Therefore, working with predefined and more classic shifts may strongly increase the difficulty of finding a complete assignment. Consequently, we choose to work with a precision of one minute for both the task assignment and the shift design.

# 3 Two-phase approach

In this section, we precisely describe our two-phase approach. The first phase is dedicated to the building and the assignment of shifts to workers, which gives a *pattern*. The second phase is devoted to the corresponding task assignment problem. We explain the complete approach from bottom to top: We first explain in Section 3.1 how we assign tasks to workers, when the pattern is given. Sections 3.2 and 3.3 are then dedicated to neighborhoods on patterns and the initial solution. Finally, in Section 3.4 we describe the overall structure of our approach.

## 3.1 Assignment of tasks

In this section, we propose a heuristic for the subproblem in which the pattern is given, i.e., shifts are already defined. We hence have to solve a Fixed Job Scheduling Problem (FJSP) where the objective function is the equity, as defined in Section 2.2. A very important step in our algorithm consists in choosing which worker is assigned to a given task. That is why we use the notion of *criticality*, that indicates (a priori) the interest of assigning a given task to a given worker. Several versions of this notion are proposed in Section 3.1.1, while the assignment heuristic is detailed in Section 3.1.2.

### 3.1.1 Criticality

We use three types of criticality, according to the time interval we are interested in:

- The (global) *criticality* of a worker $w$ as defined for the first time in [2]. The criticality of a worker is an evaluation of the way a worker will be required and it is hence high when a worker has skills and availabilities to do most of the tasks. It is a ratio between the overall amount of work an employee is able to do and the time he can spend to that work.
- The *local criticality* $c(w,t)$ of a worker $w$ and a task $t$: It is defined as the opposite of the number of tasks that $w$ is able to do and that overlap $t$.
- The *local p-criticality* $c_p(w,t)$ of a worker $w$ and a task $t$: It is defined as the opposite of the total processing time of tasks that $w$ is able to do and that overlap $t$.

These notions play a major role in the assignment algorithm, detailed in the next section.

### 3.1.2 The algorithm

We detail here the heuristic consisting in assigning fixed tasks when a pattern is given. We iteratively consider the maximal cliques $\mathcal{K} \in \mathcal{C}$ in the incompatibility graph. For each clique (i.e., set of maximal overlapping tasks), we try to assign a worker to each task lying in the clique, by solving a maximum weighted matching problem with the hungarian algorithm (cf [10]) on a *weighted assignment graph* $G(\mathcal{K}) = (\mathcal{T} \cup \mathcal{W}, \mathcal{E})$, where $\mathcal{T}$ and $\mathcal{W}$ denotes respectively the set of tasks $\mathcal{T}_\mathcal{K}$ in clique $\mathcal{K}$ and the set of workers $\mathcal{W}_\mathcal{K}$ available and skilled for at least one of these tasks. There is an edge $(t, w) \in \mathcal{E}$ if and only if the worker $w$ may be assigned to the task $t$. The weight that we put on each edge is a heuristic indicator of our algorithm that shows the interest of assigning a given task to a given worker. After extensive computational tests, we decided to keep the four following choices:

- Method *Tar*: The remaining targeted working time $W_r(w_\mathcal{K})$ of $w_\mathcal{K}$, i.e., the difference between the targeted working time and the sum of the processing times of tasks already assigned to $w_\mathcal{K}$.
- Method *TarPr*: The difference between $W_r(w_\mathcal{K})$ and $p_r(w_\mathcal{K})$, $p_r(w_\mathcal{K})$ being the sum of the processing times of all remaining tasks that $w_\mathcal{K}$ is able to do. Note that in this measure, we use a combination of minutes and number of tasks, which is not intuitive. Nevertheless both criteria are influent that is why we did experiments with different weights (from 1 to 20) on the two data. Since the best result was obtained with equal weights, we decided to keep unitary weights.
- Method *CrTar*: The sum of the local criticality $c(w_\mathcal{K}, t_\mathcal{K})$ and $W_r(w_\mathcal{K})$.
- Method *pCrTar*: The sum of the local p-criticality $c_p(w_\mathcal{K}, t_\mathcal{K})$ and $W_r(w_\mathcal{K})$.

In order to obtain strictly positive weights, we add a large positive constant to each edge. The second parameter of our algorithm is the order in which we explore the cliques; we tested five different methods and decided to keep the three methods leading to the best results:

- Method *First*: Choose the first clique in the chronological order.
- Method *LongTight*: Choose the clique containing the longest task. If there is a tie, choose the one having the smallest difference between the number of workers and the number of tasks. If there still is a tie, choose the first one.
- Method *TightLong*: Choose the clique for which the difference between the number of workers and the number of tasks is the smallest. If there is a tie, choose the clique containing the longest task. If there still is a tie, choose the first one.

The combination of these two parameters leads to 12 different settings for the algorithm. Each setting may be used to find a solution to this subproblem.

This task assignment procedure will be called in each step of the local search in order to build an assignment for each pattern. The next part is dedicated to this pattern building.

## 3.2 Pattern neighborhood

In order to define new patterns from an old one, we use 5 different neighborhoods defined in the following:

- *Add*: Among the workers having the lowest gap between the targeted working load and the real working load, pick one and add a shift randomly, in order to cover at least one task.
- *Extend*: Among the workers having the lowest gap between the targeted working load and the real working load, pick one and extend a shift randomly as much as possible in order to respect legal constraints, either at the beginning or at the end of the shift.
- *Remove*: Among the workers having the highest gap between the targeted working load and the real working load, pick one and remove a shift randomly.
- *GuidedEquity*: Among the workers having the highest gap between the targeted working load and the real working load, we pick one worker $w$ and add a shift containing a task done by $w$ to another worker.
- *GuidedRealisability*: Among the unassigned tasks, pick a task $t$. Pick a worker for whom it is possible to move a shift to cover $t$. If all the tasks are allocated, among the tasks that are covered the least(i.e., among the tasks for which the difference between the number of available workers and the number of overlapping tasks is the smallest) we pick one and add a shift containing this task.

The four first neighborhoods aim mainly at improving the equity among workers, either by increasing the shifts of a worker that does not work enough (for *Add* and *Extend*) or by reducing the amount of work done by a worker doing too many tasks (for *Remove* and *GuidedEquity*). The last neighborhood is more dedicated to the feasibility of our problem.

After having assigned tasks to workers, it is possible that some tasks remain unassigned. In such cases we process a *local descent* that slightly modifies the existing pattern. More precisely, for each unassigned task, we try to find a worker for which it is possible either to extend or to add a shift, such that this worker can be assigned to this task.

The next section is dedicated to the way of building a first feasible solution for the current problem.

## 3.3 Initial solution

### 3.3.1 Computation of an initial pattern

To find an initial solution, we first start by computing an initial pattern. It relies on the use of the criticality of a worker : The set of workers is ordered according to increasing criticality, so that we first deal with workers associated to a low criticality. Then, for a task $t \in T$, we find the first worker $w$ able to perform $t$ and we add a shift $s$ to $w$ so that $s$ includes $t$. Then, we remove from $\mathcal{T}$ all the tasks covered by shift $s$ and for which worker $w$ is skilled. We iterate while $\mathcal{T}$ is not empty. Finally, we try to cover the resource profile $p$, computed as the number of tasks $p(x)$ at each time instant $x$. More precisely, for each time $x$ where the number of workers having a shift is strictly less than the number of tasks $p(x)$, we add a shift including time $x$ to an available worker, if possible.

Once the initial pattern is computed, we use it to compute an initial assignment, as described in the next section.

### 3.3.2 Computation of initial solution

Given a pattern, we try to compute an allocation of tasks to workers, as mentioned in Section 3.1, for each possible combination of the parameters $cc$ (choice of clique) and $wc$ (choice of weight). Each method is improved in two steps (reducing shifts and doing a local descent) that are proposed and discussed in Algorithm 1 in Section 3.4. The best assignment and its pattern are chosen to initialise the method. Note that we also keep track of the best method, since we will use it for several iterations of our two-phase method.

At the end of the search, we then have the best known solution, that has to be returned.

### 3.3.3 Post-processing procedure

It appears that, on several cases, it is possible to improve the best solution by a dedicated procedure that is too time-consuming to be called at each iteration. This is why we keep this procedure as a post-processing one, introduced in the following: The post-processing procedure consists in reducing (resp. increasing) the amount of work of the worker $w_i$ having the biggest (resp. lowest) $\Delta_i$ (difference between the targeted working load and the real working load).

**Insertion method**

First, for worker $w_0$ having the biggest $\Delta_i$, we pick the longest task $t$ (done by worker $w_1$) in the set of tasks that $w_0$ is able and available to do, that verifies inequalities $\Delta_1 + p(t) \leq \max_i \Delta_i$ and $\Delta_0 - p(t) \geq \min_i \Delta_i$ (in order to avoid an increase of the objective function) and assign $t$ to $w_0$. This method is illustrated with Figure 2, where the task 4 is moved from a worker $w_1$ to $w_0$ (the one having the biggest $\Delta_i$).

**Removal method**

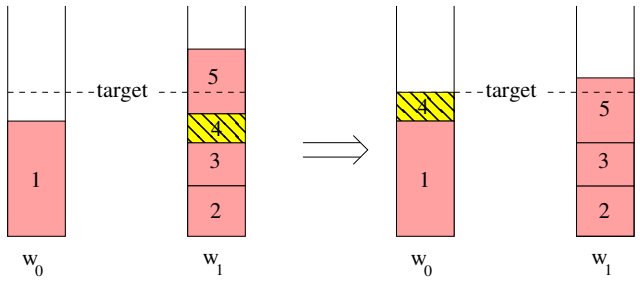


Figure 2: Post-processing procedure: insertion method: insert a task to the worker having the biggest $\Delta_i$

Second, for the worker $w_0$ having the lowest $\Delta_i$, we pick the longest task $t$ done by worker $w_0$ and for which another worker $w_1$ is available and qualified, such that $\Delta_1 - p(t) \geq \min_i \Delta_i$ and $\Delta_0 + p(t) \leq \max_i \Delta_i$ and assign $t$ to $w_1$. Figure 3 is illustrating this method.

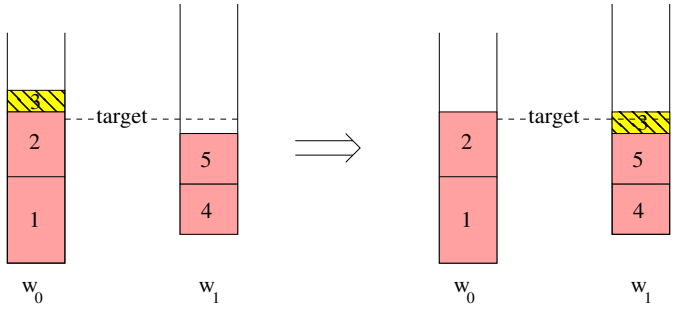


Figure 3: Post-processing procedure: removal method: remove a task from the worker having the lowest $\Delta_i$

**Swap method**

Using the same methodology, we look for a possible swap between a task $t_0$ done by the worker $w_0$ having the biggest (resp. smallest) $\Delta_i$ and a task $t_1$ done by another worker $w_1$, i.e., we assign $t_0$ to $w_1$ and $t_1$ to $w_0$, under the condition that the inequalities guaranteeing an improvement of the objective function are verified.

Note that each of the four post-processing methods is done iteratively at the end of the local search, as long as we can improve the objective function.

### 3.4 Global structure

The two-phase approach we propose consists in designing shifts in the first phase and assigning tasks to workers in the second one. We iterate between the two phases in order to improve the current solution. In this section, we describe the local search procedure that is used in the first phase, in order to improve the design of the pattern. Our local search consists in trying to improve the best known solution at each iteration, by exploring a part of a large neighborhood. Algorithm 1 describes the main steps in our local search algorithm, organised as follow:

**Lines 1-2** We compute an initial solution of our problem by first defining a pattern and then an assignment of the tasks to workers. This is discussed in detail in Section 3.3.

**Line 6** At each iteration, the first phase consists in computing new patterns by generating a set $\mathcal{P}$ of neighbors from the best pattern at the previous iteration. The different neighborhoods we propose are described in Section 3.2.

**Lines 8-9** For each generated pattern, we compute an assignment of tasks to workers. This is the second phase of our method, and it is discussed in Section 3.1. If it is not possible to assign all the tasks, we do a local descent as explained in Section 3.2.

**Lines 10-15** If necessary, we store the new local and global best solutions and patterns.

**Lines 16** In order to diversify patterns, we propose to reduce the length of the shifts by recomputing the starting and finishing time of each shift, according to the set of tasks assigned to each worker. This reduction makes the operators *Add* and *Extend* more effective, which allows to obtain very different patterns. The frequency of this operation is discussed in Section 4.1.

**Line 17** After several non-improving iterations, the strategy of the second phase may be replaced by the one that gives the best solution on the current pattern. The interest of this change is to escape local optima. The frequency of this change is discussed in Section 4.1.

**Line 18** The post-processing procedure is done at the end of the local search, in order to improve the equity of our best solution. This step is presented in Section 3.3.3.

## 4 Experiments

This section is dedicated to computational experiments validating our approach. All the tests have been done on an Intel Core i3-540 (3.06 GHz & 8G RAM)

**Algorithm 1:** Overall design of the proposed approach

```
Input: An instance instance
Output: A solution bestSolution
bestPattern ← computeInitialShifts(instance)  // cf Section 3.3.1
(bestSolution, cc*, wc*) ← computeInitialSol(bestPattern)  // cf Section 3.3.2
localBestPattern ← bestPattern
localBestSolution ← bestSolution
while (timeOut not reached) do
    P ← generateNeighborhood(localBestPattern)
    for (pattern ∈ P) do
        solution ← computeFJSPSolution(pattern, cc*, wc*)  // cf Section 3.1.2
        localDescent(pattern,solution)
        if improve(localBestSolution, solution) then
            localBestSolution ← solution
            localBestPattern ← pattern
        if improve(bestSolution, solution) then
            bestSolution ← solution
            bestPattern ← pattern
    reduceShifts(localBestPattern, localBestSolution)
    update(cc*,wc*)
PostProcessing(bestSolution,bestPattern)  // cf Section 3.3.3
return bestSolution
```

with a time limit of five minutes. In order to validate our approach, we test it on a set of 720 instances introduced in [11]. Results are compared based on four differents indicators:

- "Complete": The number of complete solutions, i.e., for which all tasks are assigned.
- "Inequity": The mean inequity value of best solutions (over complete solutions only).
- "Assigned": The average percentage of assigned tasks (over partial solutions only).
- "Time": The mean CPU time of best solutions, in seconds.

## 4.1 Parameters design

Extensive computational tests have been performed to obtain the best setting :

- The update of the strategy (cc*, wc*) in the second phase is triggered only on some iterations, based on a parameter $\lambda = 5$, 10 or 50, corresponding to the number of non-improving iterations between two updates. The smaller $\lambda$, the more it brings diversification within the second phase.

- The reduction of shifts is determined by a condition that has been tested for several settings. First, it is triggered only on some iterations, based on a parameter $\gamma = 5$, 10 or 50, corresponding to the number of iterations between two reductions. Second, it may be triggered only on solutions that are not complete yet ($\theta = 1$), or it may be triggered on all solutions ($\theta = 0$).

- The percentage, $\delta$, of eligible workers for operators *Add*, *Extend*, *Remove* and *GuidedEquity* has been tested for values 0.2, 0.6 and 1.0. The higher $\delta$, the more it brings diversification within the first phase.

Crossing these different settings leads to run our approach $54 * 720 = 38,880$ times. In order to ease the reading, we do not present the results in detail, but we give general trends on each parameter.

The inequity between workers increases with $\lambda$ by 20%, meaning that the approach benefits from an increased diversification during the second phase. Note that $\lambda$ has no clear impact on the number of complete solutions. Therefore we fix $\lambda$ to 5.

When $\gamma$ increases, if $\theta$ is set to 0, then both the inequity and the number of complete solutions tends to decrease. To obtain more complete solutions, one should consequently set $\gamma$ to a small value, but it would worsten the inequity. On the contrary, if $\theta$ is set to 1, then decreasing $\gamma$ still reduces the inequity among workers, by a factor 2, but without decreasing the number of complete solutions. In this way, the approach is able to find more complete solutions (best results obtained with $\gamma = 10$), and complete solutions have a smaller inequity.

Finally, setting $\delta$ to 1.0 slightly improves the number of complete solutions, meaning that the design of the neighborhood benefits from an increased diversification.

On the whole the best configuration is given by $\delta = 1$, $\gamma = 10$, $\theta = 1$ and $\lambda = 5$. This global setting highlights the interest of searching for very different patterns in the first phase. It also highlights the importance of solving their assignment problems with very different strategies, which justify the design of 12 different strategies. Moreover, it seems effective to decrease patterns' diversification once a complete solutions has been found, in order to keep searching with similar patterns. On the contrary, as long as some tasks remain unassigned, it seems more relevant to change the current pattern as much as possible.

In each iteration, the design of the neighborhood has also a tremendous importance. That is why we tested different settings, by changing the number of neighbors generated by each operator ( *Add*, *Extend*, *Remove*, *GuidedRealisability* and *GuidedEquity*). It turns out that we obtain the best results when 10 neighbors are generated by each of the non-guided neighborhoods (i.e., *Add*,

| Tightness | Indicator | Size | | | | |
|---|---|---|---|---|---|---|
| | | 100 | 200 | 300 | 400 | All |
| 600 | Complete | 53/54 | 59/60 | 58/58 | 59/60 | 229/232 |
| | Inequity (min) | 28 | 34 | 40 | 47 | 38 |
| | Assigned (%) | 97.6 | 99.0 | 99.7 | 99.8 | 98.3 |
| | Time (s) | 145 | 166 | 191 | 236 | 184 |
| 800 | Complete | 42/47 | 50/55 | 53/59 | 55/59 | 200/220 |
| | Inequity (min) | 35 | 35 | 38 | 44 | 38 |
| | Assigned (%) | 97.8 | 98.6 | 99.2 | 99.7 | 98.5 |
| | Time (s) | 155 | 138 | 186 | 202 | 170 |
| 1000 | Complete | 11/22 | 22/37 | 42/56 | 40/57 | 115/172 |
| | Inequity (min) | 72 | 42 | 46 | 51 | 49 |
| | Assigned (%) | 96.7 | 97.7 | 98.8 | 99.0 | 97.7 |
| | Time (s) | 167 | 156 | 173 | 196 | 173 |

Table 1: Two-phase method results(time limit: 5 min)

*Extend* and *Remove*) compared to 5 for the guided ones (i.e., *GuidedRealisability* and *GuidedEquity*). Hence, with the values retained for each parameter, we generate 40 neighbors in our approach.

## 4.2 Experimental results

Using the best parameters design, we obtained very good results that are summarised in Table 1.

Overall results are very satisfying since we are able to find a complete solution for 544 out of 624 instances (we prove that 96 instances do not admit a complete solution by using the ILP approach proposed in [12]), i.e., 87.2%. Note that there is an important disparity due to tightness: We are able to find a complete solution for all but three instances with tightness 600, and only 66.9% when tightness is equal to 1000. However, when we are not able to find a complete solution, the percentage of assigned tasks is very important, since it is more than 96%. For the objective function, the results are also very good, since we are able to deliver a mean inequity with a value less than one hour, which is satisfying from an industrial point of view. Equity deteriorates with the number of tasks and with the tightness of our instances, but it remains under one hour for 92% of instances. Note that the post-processing procedure used to improve the equity reduces the inequity by 30%.

## 4.3 Comparison with [11]

We compare our approach with the one given in [11]. Note that the hardware and software environment is the same in the two works. Many strategies are proposed in [11]; we compare our approach with two strategies: the one giving

the best results regarding to the indicator "Complete" (named MW-$\bar{l} = 180$ LN) and the one giving the best results in terms of inequity (named LW BT). Recall that there is a small modification of the constraints with respect to [11], and we hence adapt their model to have a fair comparison. Corresponding results are given respectively in Tables 2 and 3 in Appendix A.

Our approach clearly outperforms the literature. For instance, if we compare with the approach MW-$\bar{l} = 180$ LN from [11] (which gives the best results in terms of feasibility), we find 15% additional complete solutions, and we are able to reduce the objective function (i.e., improve the fairness) by about 1000 minutes. Compare to the approach LW BT from [11], dedicated to finding good solutions, we are able to find 180% additional complete solutions, with a slightly better inequity (there is a difference of 2 minutes between the two approaches).

The mean percentage of assigned tasks is also slightly better, and very close to 100%. The mean CPU time to obtain the best solution is higher in our approach, because we are able to improve our initial solution, on average, 72 times per instance. It nevertheless is quite far from the time limit of five minutes that is given by the company. On the whole, our approach performs well both on the inequity and the number of complete solutions, whatever the number of tasks or the tightness.

# 5 Conclusion and future work

In this paper, we studied the SDPTSP-E and proposed a two-phase heuristic, consisting in first designing shifts and then assigning tasks to employees, with iterations between the two phases in order to obtain solutions with a good equity. Our experimental results outperform the literature, and hence justify this two-phase approach, even if heuristic procedures are used in the two phases.

Future work may consist in studying the feasibility of our instances, in order to assess in a more accurate way the performance of different approaches. Lower bounds on the inequity is also a research avenue, but this is a very challenging study, since shifts are not fixed in this problem and hence the combinatoric is very high.

# References


[1] Alfares, H.K.: Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. Annals of Operations Research 127, 145–175 (2004)

[2] Bellenguez, O., Néron, E.: Methods for solving the multi-skill project scheduling problem. In: Proc. 9th International Workshop on Project Management and Scheduling (PMS). pp. 66–69 (2004)

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

[4] Burke, E.K., De Causmaecker, P., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. Journal of Scheduling 7, 441–499 (2004)

[5] Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1), 3–27 (2004)

[6] Ingolfsson, A., Campello, F., Wu, X., Cabral, E.: Combining integer programming and the randomization method to schedule employees. European Journal of Operational Research (EJOR) 202(1), 153–163 (2010)

[7] Kolen, A., Lenstra, J., Papadimitriou, C., Spieksma, F.: Interval Scheduling: A Survey. Naval Research Logistics 54, 530–543 (2007)

[8] Kovalyov, M., Ng, C., Cheng, T.: Fixed interval scheduling: Models, applications, computational complexity and algorithms. European Journal of Operational Research (EJOR) 178, 331–342 (2007)

[9] Krishnamoorthy, M., Ernst, A.T.: The personnel task scheduling problem. In: Optimization Methods and Applications, pp. 343–368. Springer (2001)

[10] Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2), 83–97 (1955)

[11] Lapègue, T., Bellenguez-Morineau, O., Prot, D.: A constraint-based approach for the Shift Design Personnal Task Scheduling Problem with Equity. Computers and Operations Research 40(10), 2450–2465 (2013)

[12] Lapègue, T., Bellenguez-Morineau, O., Prot, D.: Ordonnancement de personnel avec affectation de tâches: un modèle mathématique. In: Roadef'13 (2013)

[13] Lequy, Q., Desaulniers, G., Solomon, M.: A two-stage heuristic for multi-activity and task assignment to work shifts. Computers & Industrial Engineering 63(4), 831–841 (2012)

[14] Lucić, P., Teodorović, D.: Metaheuristics approach to the aircrew rostering problem. Annals of Operations Research 155, 311–338 (2007)

[15] Mabert, V.A., Watts, C.A.: A Simulation Analysis of Tour-Shift Construction Procedures. Management Science 28(5), 520–532 (1982)

[16] Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., Housos, E.: A systematic two phase approach for the nurse rostering problem. European Journal of Operational Research 219, 425–433 (2012)

# Appendix - A - Results provided in [11]

| Tightness | Indicator | Size | | | | |
|---|---|---|---|---|---|---|
| | | 100 | 200 | 300 | 400 | All |
| 600 | Complete | 53/54 | 59/60 | 58/58 | 59/60 | 229/232 |
| | Inequity (min) | 736 | 993 | 1079 | 1143 | 994 |
| | Assigned (%) | 97.6 | 99.0 | 99.7 | 99.8 | 98.3 |
| | Time (s) | 161 | 196 | 188 | 183 | 182 |
| 800 | Complete | 32/47 | 41/55 | 50/59 | 53/59 | 176/220 |
| | Inequity (min) | 732 | 1019 | 1127 | 1218 | 1057 |
| | Assigned (%) | 97.9 | 98.9 | 99.2 | 99.7 | 98.6 |
| | Time (s) | 154 | 154 | 157 | 154 | 155 |
| 1000 | Complete | 5/22 | 10/37 | 20/56 | 33/57 | 68/172 |
| | Inequity (min) | 759 | 928 | 1167 | 1281 | 1157 |
| | Assigned (%) | 95.9 | 97.6 | 99.0 | 99.1 | 97.6 |
| | Time (s) | 91 | 92 | 97 | 104 | 96 |

Table 2: Best feasibility results obtained in [11] (with strategy MW-$\bar{l} = 180$ LN)

| Tightness | Indicator | Size | | | | |
|---|---|---|---|---|---|---|
| | | 100 | 200 | 300 | 400 | All |
| 600 | Complete | 31/54 | 41/60 | 46/58 | 41/60 | 159/232 |
| | Inequity (min) | 35 | 41 | 38 | 37 | 38 |
| | Assigned (%) | 97.8 | 99.3 | 99.6 | 99.7 | 98.9 |
| | Time (s) | 60 | 36 | 19 | 27 | 35 |
| 800 | Complete | 5/47 | 5/55 | 11/59 | 12/59 | 33/220 |
| | Inequity (min) | 38 | 51 | 61 | 64 | 57 |
| | Assigned (%) | 96.9 | 98.1 | 98.8 | 99.0 | 98.2 |
| | Time (s) | 79 | 37 | 33 | 21 | 42 |
| 1000 | Complete | 1/22 | 0/37 | 0/56 | 1/57 | 2/172 |
| | Inequity (min) | 110 | - | - | 75 | 93 |
| | Assigned (%) | 92.2 | 94.9 | 97.1 | 97.6 | 95.5 |
| | Time (s) | 73 | 21 | 28 | 26 | 36 |

Table 3: Best feasibility results obtained in [11] (with strategy LW BT)