METHODS FOR THE SHIFT DESIGN AND PERSONNEL TASK SCHEDULING PROBLEM

Tanguy Lapègue, Odile Bellenguez-Morineau, Damien Prot

10^{th} *International Conference on MOdeling, Optimization and SIMlation - MOSIM14 - November 5-7-2014- Nancy - France "Toward circular Economy"*

Methods for the Shift Design and Personnel Task Scheduling Problem

T. LAPEGUE, O. BELLENGUEZ-MORINEAU, D. PROT

LUNAM Université / Ecole 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
{tanguy.lapegue, odile.morineau, damien.prot}@mines-nantes.fr

ABSTRACT: *This paper presents a large neighborhood search and compares it to the other methods used to solve the Shift Design and Personnel Task Scheduling Problem. Basically, this problem aims at designing a roster while assigning fixed tasks to an heterogeneous workforce. Such a problem may occur in several contexts, where industrial activity requires a sharp and efficient management of workers. In this paper, we focus on the particular case of a company specialized in drug evaluation. This company is facing a scheduling problem that may be modelized as a Shift Design and Personnel Task Scheduling Problem. Based on realistic instances of this particular problem, we compare the relative performances of three different methods : A two-phase method, a constraint-based approach and the large neighborhood search derived from the constraint-based model.*

KEYWORDS: *Personnel Scheduling, Shifts Design, Tasks Assignment, Equity.*

1 INTRODUCTION

In this paper, we study the SDPTSP (Shift Design and Personnel Task Scheduling Problem), initially introduced in (Lapègue, Bellenguez-Morineau & Prot 2013a). This problem consists in designing individual schedules for heterogeneous workers, which amounts to fixing days-off, designing shifts and assigning fixed tasks within these shifts. This problem shares some characteristics with other problems, such as the Nurse Rostering Problem (Burke, De Causmaecker, Berghe & Landeghem 2004) and the Tour Scheduling Problem (Mabert & Watts 1982, Alfares 2004). These problems also deal with the design of employee rosters, but they do not consider the assignment of fixed tasks that cannot be preempted. This particular point is considered in other problems such as the Fixed Job Scheduling Problem (Kolen, Lenstra, Papadimitriou & Spieksma 2007, Kovalyov, Ng & Cheng 2007) and the Personnel Task Scheduling Problem (Krishnamoorthy & Ernst 2001), but these problems do not consider the design of employee rosters. In the SDPTSP both aspects have to be taken into account, which explains the need of dedicated approaches. The first section of this paper introduces the industrial background of the real-life case study we focus on. The second section gives a formal description of the SDPTSP. Then, we present the different resolution methods, and finally compare them on a set of instances, inspired by the real-case study.

2 INDUSTRIAL BACKGROUND

New drugs have to be tested on human body in order to get a marketing authorization. These tests are performed on behalf of pharmaceutical laboratories by independent companies, certified by the Ministry of Health. The real-case study of this article is based on the scheduling problem of such a company. The testing protocol of a new drug is a dedicated procedure, very detailed, that gives a description of all the clinical tasks to be performed, along with their relative starting time and duration. These tasks are performed on volunteers, *i.e.* people who are recruited by the company and hospitalized during the study. Each week, all the tasks related to the ongoing protocols have to be assigned to qualified and available employees. The individual rosters resulting from this assignment have to respect a set of legal and organizational constraints. Moreover, they are expected to be as fair as possible.

While assigning clinical tasks, the chief nurse has to consider compulsory tasks and administrative activities because they have an impact on the availability of workers and also on the clinical workload a worker may be assigned to compulsory tasks, such as meetings and trainings, are fixed in time, already assigned to employees and cannot be preempted. Administrative activities, such as writing medical reports, are not fixed in time, may be preempted and they require a

large experience but no specific skills. Therefore, the sharing of the administrative and compulsory workloads, which is performed by the chief nurse before dealing with clinical tasks, is not well-balanced. As a consequence, the only way to get a fair roster is to balance administrative and compulsory workloads with clinical workload, which means that workers should be assigned to different clinical workloads. To capture that, we associate to each employee a *targeted clinical workload*, defined on a weekly basis by the chief nurse. Then, a roster is all the more fair than workers are close to their targeted clinical workload. Indeed, if a worker is above his/her targeted clinical workload, then he/she will have great difficulties to perform all his/her administrative activities. On the contrary, a worker that is under his/her targeted workload, will have a lot of free time. But in that case, the workload is probably not well-balanced among workers, meaning that some of them would be above their target, or simply feel having an unfair timetable.

Note that clinical tasks are fixed in time to the minute, with durations ranging from 5 minutes to 5 hours. To cover this workload, decision makers want to keep a high level of flexibility to build complete schedules more easily. This lack of fixed shifts is a strong characteristic of the problem: Employees could start and end their shifts whenever it is necessary, provided the resulting shifts and individual rosters respect the set of constraints derived from work regulation, company organization, and nurses agreements. Consequently, shifts refer to continuous working periods that may overlap two calendar days. This flexibility must be preserved because it is mandatory to cover all the clinical tasks that may arise whenever during the week. Even with such a high flexibility, it occurs that the regular workforce has to be strengthened with externals in order to be able to perform all tasks. This comes from the clinical tasks which are fixed in time and may induced a clinical workload that cannot be covered by regular workers only. In this case the decision maker tries to evaluate the needs on externals, by looking at the tasks left unassigned. Then, the required number of externals is hired for a few days so that all the clinical tasks are assigned to qualified and available workers.

The scheduling process is currently hand-performed on a weekly basis by two decision makers for a workforce of about 30 workers. This process is quite difficult to manage and it requires highly skilled decision makers with a large experience. In this context, the use of a decision-support system may ease and speed up the whole process, leading to improvements regarding workload sharing, hence a higher equity. Moreover, in case of an important work overload, such a tool may be used to minimize the amount of external workers required to cover the workload.

3 FORMAL DESCRIPTION

In the following, we define some important notions used all along the paper. A *task* is a fixed interval of work which cannot be preempted and requires a specific skill. A task is *assignable* to a worker if the worker masters the required skill and is available during the whole task. A *day* is a period of 24 hours starting every calendar days at 6 a.m, *e.g.* Monday from 6 a.m. to Tuesday 6 a.m. A *day-off* corresponds to a day without any work. A *shift* is a working time interval assigned to a worker during a day. The weekly set of shifts and days-off of a worker is called an *individual roster*. The set of all individual rosters is then simply called a *roster*. The daily set of clinical and compulsory tasks assigned to a worker during his/her shift is called a *daily schedule*. The weekly set of daily schedules and days-off of a worker is called an *individual schedule*, and the set of all schedules is called a *schedule*. The *clinical workload* of a daily schedule is given by the sum of the processing times of the corresponding clinical tasks. The *working time* of a shift is given by its duration minus the duration of the lunch break. When all the tasks are assigned to workers, the schedule is *complete*, otherwise it is *partial*. All schedules have to be *feasible*, which means that they respect constraints (O_1) to (L_7).

O_1: Employees have at most one shift per day

O_2: Employees only perform tasks assignable to them

O_3: Employees are not assigned to overlapping tasks

O_4: Compulsory assignments must be respected

O_5: Tasks cannot be preempted

O_6: Tasks cannot be assigned to more than one employee

O_7: Tasks starting in different days belong to different daily schedules

L_1: Shifts do not last more than 11 hours

L_2: Shifts are followed by a rest of at least 11 hours

L_3: Daily working times do not exceed 10 hours

L_4: Weekly working times do not exceed 48 hours

L_5: Over 7 days, employees have a break of 35 hours

L_6: Over 7 days, employees have at leat one day-off

L_7: One hour of break occurs within shifts longer than 5 hours, starting before 12 a.m. and ending after 2.30 p.m.

Without (O_7), it would be possible to build daily schedules starting in the middle of the night and ending in the middle of the morning, which is forbidden by decision-makers because it does not meet nurses agreements. About (L_7), note that this break may be splitted if required, which corresponds to the way the lunch break is currently handled by decision makers to get a complete schedule more easily.

Given this set of hard constraints, it may be impossible to find a complete schedule. In this case, the decision-maker hires additionnal externals to strengthen the regular workforce so as to be able to cover the overall workload. Of course, the decision-maker tries to hire as few as possible externals, while having acceptable schedules for regular workers. Therefore, the main goal is to minimize the number of tasks left unassigned, noted U. As a consequence, we allow partial schedules, which are useful to managers because they provide insights regarding to the need of externals. However, a complete schedule is always better than a partial one. Schedules with the same number of unassigned tasks, are distinguished based on their inequity, noted Δ. Given a set of workers $\mathcal{N}$, the inequity is defined as the difference between the highest and the lowest worker's gap value, itself being the difference Δ_n between the targeted clinical workload and the clinical workload assigned to the worker $n \in \mathcal{N}$. Based on these notations, Δ is defined as follow:

$$\Delta = \max_n(\Delta_n) - \min_n(\Delta_n)$$

.

The problem studied in this paper is refereed to as the SDPTSP||U,Δ. It consists in building an overall schedule which minimizes the number of tasks left unassigned and the inequity among workers in a lexicographique way : Given two solutions S_1 and S_2, along with their number of unassigned tasks U_1 and U_2, and their inequity Δ_1 and Δ_2, then S_1 is better than S_2 if $U_1 < U_2$ or if $U_1 = U_2$ and $\Delta_1 < \Delta_2$. Since tasks are fixed in time, a solution to this problem corresponds to an assignment of tasks to workers. This problem is addressed by three methods: A two-phase method, a constraint programming (CP) method and a large neighborhood search based on the CP model. Moreover, a local descent, noted H_Δ, is used by these methods to reduce the inequity.

4 METHODS

4.1 H_Δ: A local descent to reduce Δ

Recall that Δ depends on the highest and smallest gaps to the targeted clinical workload. As a consequence, in most cases, it depends on a few schedules only. Therefore, a good way to reduce the inequity is to focus on these schedules to try to improve them. To do so, one has to move and swap some tasks so that the clinical workload of workers who are the most below (respectively the most above) their target is increased (respectively decreased). While moving and swapping tasks, one has to consider legal and organizational constraints in order to keep feasible schedules.

We propose a local descent, noted H_Δ, which reduces the inequity in four steps repeated untill a local optimum is reached. The first step is to move tasks from workers with a big positive gap to workers with the biggest negative gap in order to increase their clinical workload. Likewise, the second step is is to move tasks from workers with a big negative gap to workers with the biggest positive gap in order to reduce their clinical workload. Steps three and four are based on the same ideas, but they perform swaps instead of simply moving one task, which may unlock some improvements. Each step is repeated as long as it reduces Δ.

All moves and swaps are performed with fixed shifts, meaning that tasks which are not fully contained in a shift cannot be assigned to the corresponding worker. Since shifts are very flexible, they may be quite small. Indeed, as a preprocessing step, H_Δ starts by computing the smallest possible shifts, which allow workers to perform the tasks assigned to them. Consequently, after this reduction, H_Δ extends all shifts to their maximum, which depends on legal constraints. This extension increases the number of considered moves, which brings more flexibility to H_Δ. The initial reduction allows Moreover, to speed up the descent, we compute for each worker the set of tasks that may be assigned to him/her, based on his/her extended shifts and skills. Then, for each worker, only moves involving tasks from this subset of feasible tasks are considered.

From a general point of view, H_Δ presents several interesting properties. First, it does not lead to the optimum of Δ. Second, applying H_Δ several times to its own output, may reduce Δ even more. Indeed, during its initialization, H_Δ starts by extended shifts randomly. Therefore, applying H_Δ several times may unlock some improvements due to changes on shifts during the preprocessing step. Finally, H_Δ does not maintain the ordering of solutions. Given two solutions S_1 and S_2, along with their inequity Δ_1 and Δ_2, such that $\Delta_1 < \Delta_2$. After using H_Δ on these solutions, we obtain new solutions, noted $S_1^{'}$ and $S_2^{'}$, along with their inequity $\Delta_1^{'}$ and $\Delta_2^{'}$. Then, we do not necessarily have $\Delta_1^{'} < \Delta_2^{'}$.

4.2 TWO-PHASE METHOD

A two-phase method has been introduced in (Prot, Lapègue & Bellenguez-Morineau 2014) to solve the SDPTSP. Recall that two-phase methods are a common way of handling problems that mixes different kinds of decisions, such as the design of shifts and the assignment of tasks. The method introduced in (Prot et al. 2014) starts by computing a set of interesting rosters (Phase 1), then, each roster is used to build a schedule by assigning tasks to workers (Phase 2). Roughly speaking, one may say that the first phase handles legal constraints, while the second one handles organizational constraints leading to

simpler problems. The interest of considering simpler problems is to be able to iterate quickly on the two phases so as to find improving solutions. At the end of each iteration, the roster leading to the best local schedule is used to generate a new set of interesting rosters, based on neighborhood operators. During the resolution, shifts tend to increase to cover tasks that were previously not well covered. However, after a few iterations, shifts tend to be too big to be extended, which reduces the flexibility of the roster and the ability of the method to build interesting rosters. To handle this problem, after a few iterations, all shifts are reduced so as to contain only the tasks within their corresponding daily schedule. Finally, when the time limit is reached, H_Δ is used to reduce the inequity. The overall design of the approach is given at the Algorithm 1.

Several neighborhood operators have been used in order to avoid being locked in local optima. Some of these operators such as the operators *Add*, *Remove* and *Extend* are quite natural. Indeed, they respectively add, remove and extend one shift. Operators *Add* and *Extend* select one worker among a subset of workers that were not working enough in the previous solution (according to the *targeted clinical workload* indicator), and they try to add a new shift to this worker or to extend an existing one. Workers that were not working enough in the previous solution are favored by these operators, so that it will be easier in the second phase to increase the clinical workload of these workers. On the contrary, the operator *Remove* focuses on workers that were working too much in the previous solution. Besides, two dedicated operators, called *Cover* and *Equity*, are also used to improve respectively U and Δ. Note that each operator is used several times on the previous best roster in order to obtain a neighborhood of the previous roster.

The *Cover* operator selects a task t among the set of tasks that were not well covered in the previous solution. For instance, if some tasks are unassigned, then t will be one of these tasks. On the contrary, if all tasks are assigned, then t will be chosen among the tasks which are covered by the smallest number of shifts. Once t is selected, the *Cover* operator tries to add a shift to a worker so as to increase the number of shifts covering t. In this way, some of the tasks that were not assigned or difficult to assign will be easier to handle, which aims at easing the building of complete schedules.

The *Equity* operator selects a worker w_1 among a subset of workers that were working too much in the previous schedule. Then, it tries to add a shift to a worker $w_2 \neq w_1$, so that the new shift contains at least one task that was assigned to w_1. In this way, some of the tasks that were assigned to a worker working too much will be easier to assign to other workers, leading to reduce the inequity.

The second phase of the method, which consists in assigning tasks to workers based on a given roster, is addressed by a heuristic that successively assigns a subset of tasks to workers, until no tasks may be added to the assignment. Each subset of tasks is chosen among the set of maximal sets of overlapping tasks and is consequently a maximal clique in the incompatibility graph. Consequently, tasks of the same clique have to be assigned to different workers (L_7). Therefore, the problem amounts to maximizing the matching between the set of tasks in the current clique and the set of skilled and available workers. Moreover, to favor assignments that seem promising, we add different weights on the edges of the assignment graph. Those weights are computed by heuristics in order to evaluate, a priori, how much a person could be critical or not for the tasks assignment. Therefore, the problem amounts to solving a maximum weighted matching problem, with the hungarian algorithm (Kuhn 1955).

Computational experiments lead to keep four different ways of computing weights and three different ways of sorting cliques, which leads to twelve different settings for the second phase. Weights computing is mainly based on the remaining targeted workload of workers, along with their local criticality. The remaining targeted workload of a worker w is defined as the difference between the targeted clinical workload of w and the clinical workload already assigned to w. The higher the remaining targeted workload, the higher the interest of edges linked to w. The local criticality of a worker w, depending on a task t, is defined as the opposite of the number of tasks that w is able to do and that overlap t. It may also be weighted by the processing times of these tasks. The lower the criticality, the higher the interest of the edge between w and t. Maximal cliques of overlapping tasks may be sorted by chronological order, by decreasing task duration (the clique containing the longest task comes first) or by increasing task covering (the clique containing the task that is covered by the smallest number of shifts comes first).

During one iteration, only one of the twelve settings of the second-phase is used. The choice of this setting is performed while computing the initial solution. However, after several non-improving iterations, all settings are tested once again to replace the current one by the one that performs best. In order to compute the initial schedule, we use a heuristic that takes into account workers skills and availabilities, but also the workload profile, so as to place shifts were they are required. Once the initial roster is designed, the second-phase method is run once for each setting in order to get the best possible initial solution.

Algorithm 1: Two-Phase Method

Input: An instance, noted $\mathcal{I}$,
Best and local best rosters, noted $\mathcal{R}_b$ and $\mathcal{R}_{bl}$
Best and local best schedules, noted $\mathcal{S}_b$ and $\mathcal{S}_{bl}$
Choice of cliques and weights, noted C_c^* and C_w^*
Output: The best schedule ($\mathcal{S}_b$)

```
(R_b, R_bl) ← initializeRoster(I)
(S_b, S_bl, C*_c, C*_w) ← initializeSchedule(R_b)
while ( time limit not reached ) do
    N_R ← generateNeighborhood(R_bl)
    for (R ∈ N_R) do
        S ← buildSchedule(R, C*_c, C*_w)
        if (S better than S_bl) then
            (S_bl, R_bl) ← (S, R)
            if (S better than S_b) then
                (S_b, R_b) ← (S, R)
    reduceShifts(R_bl, S_bl)
    update(C*_c, C*_w)
S_b ← H_Δ(S_b)
return S_b
```

4.3 CONSTRAINT-BASED APPROACH

A Constraint Programming (CP) method has been introduced in (Lapègue et al. 2013a) to solve the SDPTSP. Recall that CP is a powerful programming paradigm wherein relations between variables are stated through constraints. It is based on the exploration of a search tree that is truncated during the search by the filtering algorithms of constraints. The approach introduced in (Lapègue et al. 2013a) relies on a CP model along with dedicated search strategies. Once the time limit is reached, H_Δ is used to reduce the inequity.

The overall idea is to solve simultaneously the problems of assigning a set of tasks $\mathcal{T}$ to a set of workers $\mathcal{N}$, so that the resulting schedule respect legal and organizational constraints. Decision variables are given by a matrix of set variables (a variable that represents a set) which gives for each worker $n \in \mathcal{N}$, and for each day $d \in \mathcal{D}$, the set of tasks $\mathcal{T}_{d,n} \subset \mathcal{T}$ performed by worker n during day d, *i.e.* the daily schedule of worker n. These variables are also used to obtain the matrix of shifts, which gives for each worker $n \in \mathcal{N}$, and for each day $d \in \mathcal{D}$, the start $Fi_{d,n}$ and the end $La_{d,n}$ of the corresponding shift. The start (respectively the end) of a shift of a worker is given by an integer variable which corresponds to the earliest start (respectively the latest end) among the tasks of the corresponding daily schedule. The link between $\mathcal{T}_{d,n}$, $Fi_{d,n}$ and $La_{d,n}$ is ensured by constraints *minOverSet* and *maxOverSet*. Based on $Fi_{d,n}$ and $La_{d,n}$ variables, legal constraints related to maximal working time and minimal rest duration are easily stated through arithmetic constraints. If a variable $\mathcal{T}_{d,n}$ is empty, then $Fi_{d,n}$ and $La_{d,n}$ are instantiated to special values that guarantee the consistency of the model. Note that variables $\mathcal{T}_{d,n}$ correspond to a partition of $\mathcal{T}$, which is specified through the constraint partition. In this way, most constraints related to the assignment of tasks are respected.

To prevent employees from working simultaneously on two tasks, the first step is to compute the set of maximal sets of overlapping tasks $\mathcal{C}$ in pre-processing. Actually, it amounts to finding all maximal cliques in the corresponding incompatibility graph. Then, for each clique $\mathcal{K} \in \mathcal{C}$, it is possible to state that the intersection between $\mathcal{K}$ and any $\mathcal{T}_{d,n}$ contains at most one element. Remaining constraints related to skills requirements and workers availabilities are handled during pre-processing. A dummy worker is added to the workforce in order to allow partial schedules: All the tasks can be assigned to the dummy worker without any constraint violation. Consequently, the number of unassigned tasks correspond to the number of tasks assigned to the dummy worker. Based on variables $\mathcal{T}_{d,n}$ the clinical workload of each worker is easily computed with the constraint *sumOverSet*. Once the clinical workload of each worker is known, the inequity objective is easily obtained through arithmetic constraints.

Several search strategies designed to find fair schedules or complete schedules are proposed in (Lapègue et al. 2013a). The strategy that gives the best results regarding the inequity of schedules is called LW-BT, which stands for "Less Working, Biggest Tasks": At each branching node, it selects the worker who is currently the most under his/her targeted clinical workload and assigns to this worker the longest task he/she may perform. One can easily see that LW-BT tries to reduce the inequity among workers at each branching node. The strategy that finds the highest number of complete schedules is called MW-LN, which stands for "Most Working, Lack of Nurse". At each branching node, it selects the worker who is currently the most above his/her targeted clinical workload and assigns to this worker the task that may be assigned to the smallest number of workers. One can easily see that MW-LN tries to handle tasks that require rare skills at the beginning of the search, but also, it tries to keep room for further assignments by assigning tasks to workers who are already quite buzy. In this way, the inequity among workers will be high, but complete schedules will be find more easily.

4.4 LARGE NEIGHBORHOOD SEARCH

The global idea of the Large Neighborhood Search (LNS) method dedicated to the SDPTSP has been presented in (Lapègue, Bellenguez-Morineau & Prot 2013b). Recall that LNS methods iterate on two main steps. Given a schedule, it first consists in a partial destruction of this schedule, followed by a repair

step that consists in exploring the resulting neighborhood in order to find improving schedules. The LNS proposed in (Lapègue et al. 2013b) relies on the CP model explained in the previous section. More precisely, both the initial schedule and the repair step are handled by the CP model. The use of an exact method to repair the current schedule allows to find local optima in each neighborhood. However, in order to be able to explore quickly several neighborhoods, the exploration of each neighborhood is bounded. Since the SDPTSP is composed of a tasks assignment problem along with a shift design problem, it is natural to consider two different neighborhoods, one that focuses more on the assignment part, and one that focuses more on the design of shifts. Therefore, the proposed LNS uses two dedicated neighborhoods. The overall design of the approach is given at the Algorithm 2.

The first neighborhood operator, noted T_{op}, selects randomly an initial task $t \in \mathcal{T}$ and destroys the assignment of all the tasks that are simultaneous or closed to t. This neighborhood destroys at least all assignments of tasks that overlap t, but the neighborhood may be even larger, depending on its current size. When working on a partial schedule, t is preferably selected among unassigned tasks.

The second neighborhood operator, called P_{op}, selects randomly a set of workers $\mathcal{W}$, depending on its current size, and destroys all assignments related to these workers. When working on a complete schedule, $\mathcal{W}$ is preferably composed of workers who are the furthest from their targeted clinical workload.

Note that both operators rely on a parameter that gives its size. Depending on the chosen neighborhood, the size is not defined in the same way. When considering the T_{op} neighborhood, it corresponds to the number of assignments that are going to be destroyed. On the contrary, when considering the P_{op} neighborhood, it corresponds to the number of workers whose assignments are going to be destroyed. The T_{op} neighborhood allows to change completely the assignment of tasks during a particular time interval, whereas the P_{op} neighborhood allows to change completely the shifts performed by a subset of workers. After a given number of non improving iterations, their sizes are increased.

At each iteration of the LNS, a large part of the schedule is frozen, which means that search strategies presented in the previous section may be too deterministic to be able to find different schedules. To handle this problem, the strategy used in the LNS is a generic CP strategy that relies highly on random branching and exploration, combined with a restart criterion. Moreover, U and Δ are handled in two different ways. The exploration of the neighborhood is designed to find new schedules with a number of unassigned tasks at least as good as the previous solution, regardless of the equity. Therefore, U is minimized, but not strictly, while Δ is not constrained. Potentially, it enables to find all optima on U, which enables to find several solutions with the same number of unassigned tasks, but with different values of Δ. This feature is interesting, because H_Δ may lead to good results even when Δ is not very good. Thus, H_Δ is used at each iteration to reduce the inequity of complete schedules. On the whole, this exploration strategy has two different goals: First it improves the number of unassigned tasks by filtering worst schedules, second it finds many different complete schedules, each one being improved by H_Δ. Therefore, it benefits from two different strengths of CP, being its ability to filter based on strong global constraints and its ability to find feasible schedules. One may wonder why non improving schedules are kept during the second step, whereas they are filtered during the first step. This comes from two facts: First the filtering of CP model on U is more efficient that the one on Δ, second the local search is very effective even on schedules that have a high inequity. Note that neighborhoods are computed on best schedules after the use of H_Δ. Therefore, the LNS is able to converge on improving schedules.

Algorithm 2: LNS Method

Input: An instance, noted $\mathcal{I}$
Current and best schedules, noted S_c and S_b
Time neighborhood operator, noted T_{op}
Personnel neighborhood operator, noted P_{op}
Current neighborhood operator, noted op
Size of op, noted size

$(S_c, S_b) \leftarrow$ initializeSchedule($\mathcal{I}$)
while (*time limit not reached*) **do**
  op $\leftarrow$ select(T_{op}, P_{op})
  $S_c \leftarrow$ destroy(S_b, op, size)
  $S_c \leftarrow$ repair(S_c)
  $S_c \leftarrow H_\Delta(S_c)$
  counter $\leftarrow$ counter $+ 1$
  if (S_c *better than* S_b) **then**
    $S_b \leftarrow S_c$
    counter $\leftarrow 0$
  size $\leftarrow$ updateSizeOfNeighborhoods(counter, size)

5 EXPERIMENTS

The three methods mentionned in this paper can be applied to solve the SDPTSP. In order to compare these methods, extensive experiments have been performed on instances inspired by the industrial case study. This section is dedicated to computational experiments validating each approach, and comparing their relative performances. All tests have been performed on an Intel Core i3-540 (3.06 GHz & 8G RAM)

with a time limit of five minutes. Tests are performed on a set of 720 instances introduced in (Lapègue et al. 2013a). Each set of 30 instances corresponds to a given combination of a number of tasks, a tightness, and a skills setting. The number of tasks is taken from 100 to 400. The tightness, which evaluates the average clinical workload of workers, is considered from 600 minutes to 1000 minutes, knowing that the common tightness for a classical week is around 800 minutes. Two skills settings are considered: The common skills setting, *i.e.* skills are mastered by most of the nurses, and the common and rare skills setting. Within this last setting, 5% of the skills are considered to be rare and thus are mastered by only 20% of the nurses, as opposed to 80% for the common ones. Repartition and length of the tasks all along the week is done to follow activity peaks in the company.

Results are compared with the following indicators:
C: **C**omplete schedules.
I: **I**nequity, in minutes, of complete schedules.
A: Percentage of **A**ssigned tasks, over partial schedules.
T: CPU **T**ime, in seconds, of best schedules.

Note that 102 instances are known to admit no complete schedules. Therefore, the number of complete schedules is given over the number of instances that may admit a complete schedule. One may wonder why we consider these instances within our tests. Actually, they are very interesting because they allow to evaluate the ability of methods to reach a schedule as complete as possible, which may be used to evaluate sharply the number of required externals. Results are given in appendices 1 to 4.

Regarding the number of complete schedules, the LNS performs better (581), followed by the two-phase method (544). The constraint-based approach comes last with respectively 473 and 194 complete schedule for strategies MW-LN and LW-BT. Note also that all methods encounter difficulties to find complete schedules when the tightness of instances is high. Moreover, among instances with a high tightness, those who have 100 tasks are the most difficult ones to solve. Regarding the average inequity, LW-BT performs best (29), followed by the LNS (27), the two-phase method (40) and finally MW-LN (130). Note also that instances with a small number of tasks are generally solved in a better way regarding to the average inequity. On the whole, the LNS provides very good schedules regarding both the number of complete schedules and the average inequity among workers. The constraint-based approach, which is the only exact method tested in this comparison does not compete with the LNS or the two-step method.

6 CONCLUSIONS

We have compared three methods, which solves the SDPTSP in different ways : First, a two-phase method, which relies on a classical decomposition, second, a constraint-based approach, which handles simultaneously the design of rosters and the assignment of tasks, and finally, a large neighborhood search based on this constraint-based approach. The constraint-based approach is able to quickly find good schedules, but it encounters difficulties to improve them, because the flexibility policy regarding to the design of shifts leads to a very combinatorial problem. On the contrary, the large neighborhood search benefits from the filtering of the constraint model along with an enhanced ability to explore different parts of the search space, leading to quick improvements of initial solutions. The two-phase method is also very effective to improve initial solutions, but since it works in two steps, some schedules may be very difficult to build. Therefore, the method leads to very good results, but it does not compete with the large neighborhood, which handles the building of rosters and the assignment of tasks simultaneously, even if only a small part of the problem is considered. Finally, the use of a local descent enables to reduce the inequity of schedules. This descent is very effective because well adapted to the problem. Indeed, since administrative activities are not scheduled, most schedules are not very dense, making it easier to move and swap tasks. To benefit from this descent, the most effective way seems to generate many schedules so as to diversify the schedules which are then improved with the descent. This idea turned out to be quite effective on the SDPTSP, but it may also be adapt to other problems.

To solve the SDPTSP||U,Δ effectively, it is a priori better to find the optimum on U before looking at Δ. In this way, we do not have to distinguish schedules that are not optimal regarding to U, leading to a smaller search space. However, proving optimality on U without a good lower bound is very difficult. Therefore, future work may focus on finding good lower bounds on the number of unassigned tasks.

References

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

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

Kolen, A., Lenstra, J., Papadimitriou, C. & Spieksma, F. (2007). Interval Scheduling: A Sur-

vey, *Naval Research Logistics* **54**: 530–543.

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

Krishnamoorthy, M. & Ernst, A. T. (2001). The personnel task scheduling problem, *Optimization Methods and Applications*, Springer, pp. 343–368.

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

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

Lapègue, T., Bellenguez-Morineau, O. & Prot, D. (2013b). A large neighborhood search for the shift design personnel task scheduling problem with equity, *Multidisciplinary International Conference on Scheduling: Theory and Applications*.

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

Prot, D., Lapègue, T. & Bellenguez-Morineau, O. (2014). A two-phase method for the shift design and personnel task scheduling problem with equity objective, *Technical report*, Ecole des Mines de Nantes, Technical Report 14/2/AUTO.

APPENDIX 1

Size	Crit.	Tightness			
		600	800	1 000	Total
100	C	53/54	33/47	5/21	91/122
	I	95	151	216	122
	A	97,71	97,89	96,05	96,74
	T	161	152	96	136
200	C	59/60	42/55	9/35	110/150
	I	95	144	192	122
	A	99,00	98,78	97,71	98,00
	T	171	142	95	136
300	C	58/58	50/58	21/56	129/172
	I	89	131	261	133
	A	99,67	99,23	98,92	99,01
	T	201	178	99	159
400	C	59/59	54/59	34/56	147/174
	I	83	111	256	133
	A	99,75	99,63	99,07	99,19
	T	171	171	129	157
Total	C	229/231	179/219	69/168	**477/618**
	I	90	132	246	**128**
	A	98,37	98,54	97,66	**97,91**
	T	176	161	105	**147**

Table 1: CP-MW-LN-H_{Δ}

APPENDIX 2

Size	Crit.	Tightness			
		600	800	1 000	Total
100	C	31/54	5/47	1/21	37/122
	I	26	34	46	28
	A	97,76	96,89	92,22	95,14
	T	55	73	75	68
200	C	41/60	5/55	0/35	46/150
	I	29	32	-	30
	A	99,26	98,05	94,92	96,82
	T	34	26	29	30
300	C	46/58	11/58	0/56	57/172
	I	29	34	-	30
	A	99,57	98,82	97,13	98,08
	T	25	25	32	27
400	C	41/59	12/59	1/56	54/174
	I	25	40	30	29
	A	99,68	99,03	97,60	98,46
	T	21	22	25	23
Total	C	159/231	33/219	2/172	**194/618**
	I	28	36	38	**29**
	A	98,88	98,15	95,47	**97,05**
	T	34	36	40	**37**

Table 2: CP-LW-BT-H_{Δ}

APPENDIX 3

Size	Crit.	Tightness			
		600	800	1 000	Total
100	C	53/54	42/47	11/21	106/122
	I	28	35	72	35
	A	97,50	97,78	96,71	97,05
	T	145	155	167	156
200	C	59/60	50/55	22/35	131/150
	I	34	35	42	36
	A	99,00	98,60	97,72	97,93
	T	166	138	156	154
300	C	58/58	53/58	42/56	153/172
	I	40	38	46	41
	A	99,67	99,19	98,76	98,94
	T	191	186	174	184
400	C	59/59	55/59	40/56	154/174
	I	47	44	51	47
	A	99,75	99,70	99,01	99,17
	T	236	202	196	211
Total	C	229/231	200/219	115/168	**544/618**
	I	38	38	49	**40**
	A	98,28	98,47	97,68	**97,90**
	T	184	170	173	**176**

Table 3: Two-phase method results

APPENDIX 4

Size	Crit.	Tightness			
		600	800	1 000	Total
100	C	54/54	46/47	20/21	120/122
	I	17	14	17	16
	A	97,67	98,07	97,10	97,38
	T	144	117	128	127
200	C	59/60	53/55	29/35	141/150
	I	27	24	26	26
	A	99,00	98,71	98,90	98,13
	T	73	175	162	162
300	C	58/58	54/58	47/56	159/172
	I	33	26	29	29
	A	99,67	99,33	98,90	99,10
	T	180	170	171	172
400	C	59/59	58/59	44/56	161/174
	I	34	31	38	34
	A	99,75	99,75	99,27	99,34
	T	278	289	195	209
Total	C	230/231	211/219	139/168	**581/618**
	I	28	24	30	**27**
	A	98,41	98,60	97,95	**98,12**
	T	158	154	155	**155**

Table 4: LNS results