A rotation-based branch-and-price approach for the nurse scheduling problem

Antoine Legrain, Jérémy Omer, Samuel Rosat

A rotation-based branch-and-price approach for the nurse scheduling problem

Antoine Legrain[a,b], Jérémy Omer[c,d], Samuel Rosat[a,e]

[a]*Polytechnique Montreal, 2900, boulevard Édouard-Montpetit, Campus de l'Université de Montréal, 2500, chemin de Polytechnique, Montreal (QC) H3T 1J4, Canada, www.polymtl.ca*
[b]*Interuniversity Research Center on Enterprise Networks, Logistics and Transportation (CIRRELT), Université de Montréal, Pavillon André-Aisenstadt, CP 6128, Succursale Centre-Ville, Montreal (QC) H3C 3J7, Canada, www.cirrelt.ca*
[c]*Institut National de Sciences Appliquées (INSA), 20, avenue des Buttes de Coësmes, 35708 Rennes, France, http://www.insa-rennes.fr*
[d]*Institut de Recherches Mathématiques de Rennes, 35000 Rennes, France, http://irmar.univ-rennes1.fr/*
[e]*Group for Research in Decision Analysis (GERAD), HEC Montréal, 3000 ch. de la Côte-Sainte-Catherine, Montreal (QC) H3T 2A7, Canada, www.gerad.ca*

Abstract

In this paper, we describe an algorithm for the personalized nurse scheduling problem. We focus on the deterministic counterpart of the specific problem that has been described in the second international nurse rostering competition. One specificity of this version of the problem is that most constraints are soft, meaning that they can be violated at the price of a penalty. The feasible space is thus much larger, which involves much more difficulty to find the optimal solution. We model the problem as a an integer program (IP) that we solve using a branch-and-price procedure. This model is, to the best of our knowledge, comparable to no other from the literature, since each column of the IP corresponds to a rotation, i.e., a sequence of consecutive worked days for a nurse, and not to a complete individual roster. We tackle instances involving up to 120 nurses and 4 shifts over an 8-weeks horizon by embedding the branch-and-price in a large-neighborhood-search framework. Initial solutions of the large-neighborhood search are found by a rolling-horizon algorithm, well-suited to the rotation model.

Keywords: nurse scheduling problem, column-generation, decomposition, branch-and-price, large-neighborhood search, rolling horizon

1. Introduction

1.1. The nurse scheduling problem

For several years, hospitals have been facing increasing shortages in most western countries, either in terms of finance or resources: beds, nurses, etc. Hence, they are often unable to provide the expected level of service. These difficulties can be tackled in two different ways: either increase, or make better use of the available resources. Given the slow increase of the budget allocated to healthcare, hospitals have no other choice than improving the use of their resources. Among the main challenges is the shortage of nurses.

We tackle here the nurse scheduling problem (NSP), i.e., the design of the schedule of a nursing service so as to satisfy the needs of the hospital and all operational constraints at minimal cost. In the NSP, the cost of a schedule is the sum of penalties such as extra-hours, under-coverage, etc. A fair deal of the complexity of the NSP can be attributed to the personalized aspect of the rosters: the same schedule may indeed have very different costs if given to one nurse or another, because they have different contracts or preferences on

Email addresses: antoine.legrain@polymtl.ca (Antoine Legrain), jeremy.omer@insa-rennes.fr (Jérémy Omer), samuel.rosat@polymtl.ca (Samuel Rosat)

days off. The NSP has been widely studied for more than two decades. The reader is referred to [6] for a review on models and solution methods. Among the numerous approaches, we focus here on those based on integer programming techniques.

In most integer programs (IPs) that model the NSP, the columns of the constraint matrix correspond to individual rosters, and the linear constraints to global requirements such as minimal staffing levels. Most constraints are therefore partitioning equalities (or covering inequalities), each of them ensuring that a given task is performed by a sufficient number of nurses. Due to the length of the planning horizon and the number of different tasks that may be performed by each nurse, it is impossible to enumerate all columns beforehand and solve the resulting IP with integer programming algorithms such as branch-and-bound. To palliate this numerical obstacle, one restricts the number of columns in the problem at first and, at each node of the branch-and-bound tree, the linear relaxation is solved by column generation; the resulting algorithm is called *branch-and-price*. For more details on the implementation of branch-and-price techniques, the reader is referred to [2]. In scheduling applications, the generation of new columns is usually based on the solution of a shortest path problem with resource constraints (SPPRC); its solution either yields a new column of negative reduced cost or asserts that the optimal solution of the restricted linear relaxation is optimal for the nonrestricted version. The reader can find an overview of column-generation algorithms in [8] and a comprehensive review on SPPRC in [14].

To the best of our knowledge, the first published attempt to tackle the NSP by means of branch-and-price dates back to 1998 [15]. The authors develop a generic framework that handles a large variety of families constraints. New columns are generated by solving a SPPRC that involves up to seven different constraints. In [1], the authors stress the personalized facet of the problem. New schedules are generated by a swap heuristic and their feasibility is checked a posteriori (whereas SPPRC generally only produces feasible schedules). The authors of [16] use a two-phase algorithm for the roster-generation procedure: they only solve the SPPRC when a heuristic procedure fails to find columns of negative reduced cost. They also undertake a detailed study of 15+ different branching rules on one or several variables. The pricing subproblem can also be solved with constraint programming, as in [13]. Finally, in [5], columns are generated by solving a SPPRC with an innovative heuristic based on dynamic programming (the heuristic part can however be deactivated for an exact solution). They also stress the importance of dual variables stabilization techniques to reduce the number of calls to the subproblem. Among other important works is [3] where the authors treat nurse and surgery scheduling by branch-and-price in an integrated approach.

1.2. Main contributions

In this article we deal with a static version of the NSP described in the second international nurses rostering competition (INRC–II). The methods we implement are motivated by the following two properties of this version of the NSP. First, every nurse is different, because he/she has his/her own past planning, preferences for days off, and work contract. This would make most techniques based on an aggregation of similar nurses inefficient. What is more, there is only a small part of hard constraints. The remaining constraints can all be violated at the price of a penalty. As a consequence, it is relatively easy to find feasible schedules, but very difficult to find an optimal one (and prove that it indeed is optimal). We thus developed a solution approach that takes these specificities into account.

Modeling approach. All previously-mentioned works that solve the NSP with branch-and-price algorithms share the same modeling approach: the pricing subproblem generates complete rosters and decision variables indicate wether these complete rosters are to be used in the solution. In our work, we do not generate complete rosters but *partial* rosters, that are called *rotations* (this denomination comes from the transportation industry where similar modelling is commonly used, see [11]). A rotation describes a sequence of worked shifts on consecutive days that should be preceded and followed by at least one rest day; it does not specify the skill used by the nurse on each shift. The IP then builds complete schedules for the nurses from the existing rotations, and allocates the skills that they must perform. Personalized rotations are then generated by solving a SPPRC for each nurse.

To the best of our knowledge, it is the first attempt to address the NSP with a rotation-based model. Our motivation for choosing this approach is to counter the effect of soft constraints by reducing the size of the search space and decrease the complexity of the subproblems. This change of paradigm also led to the development of new branching rules (Section 4.3) and to structural modifications of the network of the pricing step (Section 4.2). The separation of the allocations of shifts and skills is independent from the partial-roster approach. This decomposition, as well as the flow-model that we introduce to solve the skills allocation problem (Section 3.3) are also part of the methodological contributions of this work.

Large-neighborhood search and rolling horizon. In the aim of solving large instances, we embed the branch-and-price procedure within an adaptive large neighborhood search procedure (ALNS) described in Section 5. The ALNS is a local search where the neighborhood of a feasible solution is obtained by sequentially destroying and repairing a part of the solution. The repair phase is classically overdone by solving to optimality a reduced counterpart of the problem where the non-destroyed part of the solution is fixed. In our implementation of the ALNS, we either destroy the complete schedules of a limited number of nurses or partial schedules of a larger number of nurses. Local improvements of the initial solution are then realized by solving the restricted problem corresponding only to the destroyed schedules with our branch-and-price algorithm.

We also develop several primal heuristics based on our branch-and-price procedure to find the initial solution of the ALNS. Among them is a rolling-horizon method which sequentially computes weekly schedules in chronological order until the complete planning horizon is scheduled. One of the key challenges in rolling-horizon methods is the accurate estimation of the impacts of the decisions on the future. When computing the schedule of a given week, we thus include the following weeks in the problem but we relax the integrality constraints of variables corresponding to the future to accelerate the algorithm.

Implementation and numerical results. The algorithms described in this article are implemented in C++ and call only free and open third party libraries. The resulting code is publicly shared[1] for reproduction of the results, future comparisons, improvements and extensions.

Our numerical tests are all based on the instances of the INRC–II. These instances consider 30 to 120 nurses whose schedules must be computed over a planning horizon of either four or eight weeks. The demands relate to up to four different skills and each day is divided into four shifts. We conduct an experimental comparison of several initialization methods for the LNS, and study the sensitivity of the LNS to the choice of the search neighborhoods. Although we were not able to prove the optimality of the solutions found for these instances, we show that a rolling horizon method can be advantageously used to find an initial solution with an average integrality gap of 19.3%, and the LNS is able to reduce this value to an average 10.9% gap.

1.3. Organization of the paper

The remainder of this paper is organized as follows. The exact description of the NSP that our method can solve is given in Section 2. The rotation-based model of the NSP and the resulting formulation (IP) are described in Section 3. In Section 4, we briefly describe the branch-and-price algorithm and the underlying decomposition, and we specify the branching strategies and the pricing method we implemented. The LNS procedure, including the rolling-horizon method to find the initial solution and the choice of the neighborhood, is expounded in Section 5. In Section 6, we report numerical results that demonstrate the relevance of our approach. Section 7 provides concluding remarks.

2. Description of the problem and notations

The specific version of the NSP we consider is based on that proposed by Ceschia *et al.* [7] in the INRC–II. While the problem is stated as a dynamic one in [7], where only a part of the information is given at each stage of the solution process, here we consider its static version where all information is available beforehand.

We wish to compute the schedules of a set $\mathcal{N}$ of nurses over a planning horizon of M weeks (or $K = 7M$ days). The nurses can perform different skills and each day is divided into shifts. The sets of all skills and

[1] The code implementing the methods described in this article is publicly shared on the Git repository `https://github.com/jeremyomer/StaticNurseScheduler`

shifts are respectively denoted Σ and $\mathcal{S}$. For the sake of readability, indices are standardized in the following way: nurses are denoted as $i \in \mathcal{N}$, weeks as $m \in \{1 \ldots M\}$, days as $k \in \{1 \ldots K\}$, shifts as $s \in \mathcal{S}$ and skills as $\sigma \in \Sigma$. Finally, (k, s) denotes the shift s of day k, and is abusively called "shift (k, s)". All other data is summarized in Table 1.

Nurses	
L_i^-, L_i^+	min./max. total number of worked days over the planning horizon for nurse i
CD_i^-, CD_i^+	min./max. number of consecutive worked days for nurse i
CR_i^-, CR_i^+	min./max. number of consecutive rest days for nurse i
B_i	max. number of worked week-ends over the planning horizon for nurse i
β_i	1 if nurse i must work zero or both days on week-ends, 0 otherwise
Π_i	set of shifts (k, s) that nurse i wishes to have off
Shifts	
CS_s^-, CS_s^+	min./max. number of consecutive assignments on shift s
$\bar{\mathcal{F}}$	set of forbidden shift successions
Demand	
D_σ^{sk}	min. demand in nurses performing skill σ on shift (k, s)
O_σ^{sk}	optimal demand in nurses performing skill σ on shift (k, s)

Table 1: Summary of the input data.

Remark (Initial state)**.** *For practical reasons, it is necessary that the algorithm can handle an initial state, e.g., the information on the end of a previously worked time period. For the sake of clarity, we do not take it into consideration in the description of the method although our software handles it; the incumbent modifications on the models and the algorithm are straightforward and of no particular interest for the reader.*

The exhaustive enumeration of every constraint that can be found in the literature on NSPs would form a never-ending list that we do not intend to handle. We choose the set of constraints proposed by the organizers of the INRC–II in [7] for the following main reasons: (1) they all are usual constraints that nursing services face in practice and (2) they allow us to tackle the benchmark released by the organizers of this competition. This benchmark contains a huge number of instances (scenarios can be generated at will by combining weeks together), including large instances (up to 120 nurses) and enough constraints to make the instances close to industrial ones. Some constraints are *hard*, i.e., they may never be violated by a feasible solution; others are *soft*, i.e., they may be violated at the cost of a penalty. The objective function that we minimize is the sum of these penalties.

The specificity of this benchmark is that most constraints are soft. This eases the search for a feasible solution, but it makes the pursuit of optimality more difficult. The constraints and their types (hard/soft) are described in Table 2. The unit weight (i.e. the penalty) associated with a soft constraint **SX** in the objective function is denoted as c_X. For constraint **S2**, the unit weights for consecutive working days and consecutive shift are respectively denoted c_{2a} and c_{2b}.

3. Rotation-based model for the nurse rostering problem

The aim of this section is to describe the NSP as an IP whose main decision variables correspond to the choice of the rotations performed by each nurse. First, we provide some vocabulary needed in the rotation-based formulation (Section 3.1). Assuming that the rotations can be enumerated, it is necessary to build a valid sequence of rotations and rest periods that covers the entire planning horizon for each nurse, and to choose the specific skill used by each nurse on each worked shift. These two sub-problems are formulated as flow models in Sections 3.2 and 3.3. The complete model described in Section 3.4 includes these flow constraints and the soft constraints relative to the complete planning horizon (**S6** and **S7**).

	Hard constraints
H1	**Single assignment per day**: A nurse can be assigned at most one shift per day.
H2	**Under-staffing**: The number of nurses performing skill σ on shift (k, s) must be at least equal to the minimum demand D_σ^{sk}.
H3	**Shift type successions**: If $(s_1, s_2) \in \bar{\mathcal{F}}$, a nurse cannot work on shift s_1 on one day, and on shift s_2 on the next day.
H4	**Missing required skill**: A nurse can only cover the demand of a skill that he/she can perform.

	Soft constraints
S1	**Insufficient staffing for optimal coverage**: The number of nurses performing skill σ on shift (k, s) must be at least equal to the optimal demand O_σ^{sk}. Each missing nurse is penalized according to the unit weight but extra nurses above the optimal value are not considered in the cost.
S2	**Consecutive assignments**: For each nurse i, the number of consecutive assignments should be within $[\mathrm{CD}_i^-, \mathrm{CD}_i^+]$ and the number of consecutive assignments to the same shift s should be within $[\mathrm{CS}_s^-, \mathrm{CS}_s^+]$. Each extra or missing assignment is penalized by the unit weight.
S3	**Consecutive days off**: For each nurse i, the number of consecutive days off should be within $[\mathrm{CR}_i^-, \mathrm{CR}_i^+]$. Each extra or missing day off is penalized by the unit weight.
S4	**Preferences**: Each assignment of a nurse i to an undesired shift $(s, k) \in \Pi_i$ is penalized by the unit weight.
S5	**Complete week-end**: Every nurse i that has the complete weekend value set to true ($\beta_i = 1$), must work both days of the week-end or none of them. If he/she works only one of the two days Saturday or Sunday, it is penalised by the unit weight.
S6	**Total assignments**: For each nurse i, the total number of assignments (worked days) must be within $[L_i^-, L_i^+]$. The difference (in either direction), multiplied by the unit weight, is added to the objective function.
S7	**Total working week-ends**: For each nurse i, the number of week-ends with at least one assignement must be less than or equal to B_i. The number of worked week-ends over that limit multiplied by the unit weight is added to the objective function.

Table 2: Constraints handled by the software.

3.1. Rotations: definitions and notations

To formulate our mathematical model for the NSP, we first introduce some notations and vocabulary. The *roster* of a nurse is his/her schedule for the planning horizon, i.e., a list of *assignments* (day, shift and skill) that the nurse should perform during the planning horizon. A day with no assignment is a *rest* day (sometimes denoted as *day off*). Each individual roster satisfies constraints **H1**, **H3** and **H4**, and the set of all rosters satisfies **H2**. A *rotation* is a list of shifts (k, s) from the roster that are performed on consecutive days, and preceded and followed by a rest day. It is important to note that a rotation does **not** contain any information about the skills performed on these shifts. A roster is therefore a sequence of rotations, separated by nonempty rest periods, to which skills are added (see Example 1).

A rotation is *feasible* if it respects the single assignment and succession constraints **H1** and **H3**. Besides, the cost of a self-standing rotation can only cover the penalties associated with the soft constraints **S2**, **S4** and **S5**. Indeed, the penalties associated to the other soft constraints either require the nurse's schedule over the complete planning horizon (rest periods, total assignments/week-ends), or data about the complete pool of nurses (staff coverage).

Example 1. *Consider the following single-week roster:*

Day	*0*	*1*	*2*	*3*	*4*	*5*	*6*
Shift	*Early*	*Day*	*Rest*	*Rest*	*Night*	*Night*	*Rest*
Skill performed	*HN*	*HN*	-	-	*N*	*HN*	-

,

where N stands for nurse and HN for head nurse. The rotations of this roster, highlighted on the table above, are $((0, Early), (1, Day))$ *and* $((4, Night), (5, Night))$.

For $i \in \mathcal{N}$, the set of every feasible rotation for nurse i is denoted Ω_i and the cost of rotation $j \in \Omega_i$ is denoted as c_{ij}. Our model for the NSP is based on the computation of feasible rotations for each nurse. In the rest of the present section, we suppose that we can enumerate the set of all rotations Ω_i for every nurse $i \in \mathcal{N}$. In practice, rotations are generated and added to the problem iteratively, as described in Section 4.2.

Furthermore, the *type* of a nurse i, denoted t_i, is the subset of Σ that contains the skills that this nurse can perform. For any skill $\sigma \in \Sigma$, $\mathcal{N}_\sigma$ denotes the set of nurses that are able to perform σ. Similarly, for any type $t \in \mathcal{T}$, $\mathcal{N}_t$ denotes the set of nurses with the exact type t. Finally, the set of types that include skill σ is denoted as $\mathcal{T}_\sigma$.

3.2. A flow model for the creation of individual rosters

In this section, we describe how we build a roster over the complete planning horizon from the set Ω_i of feasible rotations for nurse $i \in \mathcal{N}$. We use a directed network over which the problem of building the *skill-less* roster of a given nurse i can be modeled as a single-unit flow problem. This network is called *rostering graph* of nurse i. By skill-less roster, we mean that it does not specify which skill the nurse performs on the shifts he/she works.

In essence, a vertex of this weighted graph corresponds to a day and an arc corresponds either to a rotation or to a rest period. A path from the source to the sink therefore yields a sequence of rotations separated by rest periods, hence a roster. The detail of the vertices and arcs of the rostering graph of a nurse is given on Table 3. The flow conservation constraints are given with the complete model in Section 3.4 (constraints (1b)–(1d)). The cost of a path is the sum of the penalties that can be represented as weights on the arcs. As a consequence, it aggregates the individual costs of rotations and rest periods (soft constraints **S2**, **S3**, **S4** and **S5**), but it cannot reflect the soft constraints **S6** and **S7** that deal with the complete planning horizon. Moreover, this graph deals with one specific nurse, so it does not include the linking staffing constraints. These two remaining groups must be added to the model on top of the flow constraints (see Section 3.4). An example of rostering graph is given in Figure 1 for a 7-days planning horizon and $\mathrm{CR}_i^+ = 4$.

Vertices	*source*, *sink*		One source node, one sink node.
	rest nodes		For each day k, one rest node R_{ik}.
	work nodes		For each day k, one work node W_{ik}.
Arcs	*rotation arcs*	$[x_{ij}]$	For each rotation $j \in \Omega_i$ starting on day k_s and ending on day k_e, one arc (W_{ik_e}, R_{ik_s}) with cost c_{ij} (**S2** + **S4** + **S5**) is added. When several rotations share the same starting and ending days, parallel arcs added.
	min. rest arcs	$[r_{ikl}]$	For each pair of days $(k,l) \in \{1 \dots K\}^2$ such that $k < l$ and $(l-k) \in \{1, \dots, \mathrm{CR}_i^+\}$, one arc (R_{ik}, W_{il}) is added with cost $c_3^{i(l-k)} = \max\{0, c_3 (\mathrm{CR}_i^- - (l-k))\}$ (min. consecutive days off penalty associated with **S3**).
	max.rest arcs	$[r_{ik}]$	For each day $k \in \{\mathrm{CR}_i^-, \dots, (K-1)\}$, one arc $(W_{ik}, W_{i(k+1)})$ with cost c_3 is added. These arcs are only used when the maximum consecutive number of rest days CR_i^+ is exceeded.
	artificial flow arcs		Arcs from the source to R_{i1} and W_{i1}, and arcs from R_{iK} and W_{iK} to the source are added at no cost.

Table 3: Description of the vertices and arcs of the rostering graph of nurse $i \in \mathcal{N}$. The notations indicated between brackets are the names of the corresponding flow variables used in the model of Section 3.4.

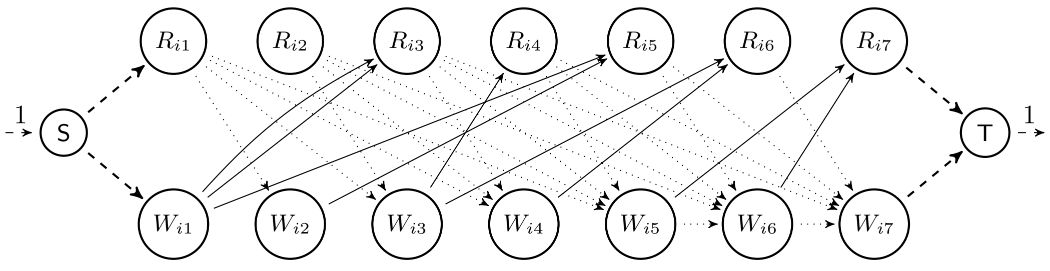


Figure 1: Example of a rostering graph for nurse $i \in \mathcal{N}$ over an horizon of $K = 7$ days, where the maximum number of consecutive rest days is $\mathrm{CR}^+ = 4$. The rotation arcs are the plain arrows, the rest arcs are the dotted arcs, and the artificial flow arcs are the dashed arrows.

3.3. A flow model for the allocation of the skills

The rostering graph does not determine the skill a nurse uses on a given worked shift. Rather than allocating them individually, for each shift (k, s) and skill σ, we determine the number $n_{t\sigma}^{sk}$ of nurses of type t that perform skill σ on shift (k, s), for all $t \in \mathcal{T}$. This information is sufficient, because the nurses with the same type that work on the same shift are locally equivalent (i.e., equivalent on this particular shift) since they can perform exactly the same skills – by definition of a type.

Recall that $\mathcal{T}_\sigma$ denotes the set of types including skill σ. Introducing $z_\sigma^{sk} \geq 0$ as the number of missing nurses to reach the optimal demand O_σ^{sk} (0 if this optimal demand is satisfied), the skill-related constraints can thus be written as follows:

H2 $\sum_{t \in \mathcal{T}_\sigma} n_{t\sigma}^{sk} \geq D_\sigma^{sk}$, $\forall s, k, \sigma$, [under-staffing]

H4 $\sum_{i \in \mathcal{N}_\sigma, j} a_{ij}^{sk} x_{ij} - \sum_{\sigma \in \Sigma \mathcal{T}_t} n_{t\sigma}^{sk} = 0$, $\forall s, k, t$, [missing required skill]

S1 $\sum_{t \in \mathcal{T}_\sigma} n_{t\sigma}^{sk} + z_\sigma^{sk} \geq O_\sigma^{sk}$, $\forall s, k, \sigma$, [optimal staffing]

where $a_{ij}^{sk} = 1$ if the rotation $j \in \Omega_i$ of nurse i covers shift (k, s) and 0 otherwise. The penalty associated with constraint **S1** is thus $c_1 \sum_{s,k,\sigma} z_\sigma^{sk}$.

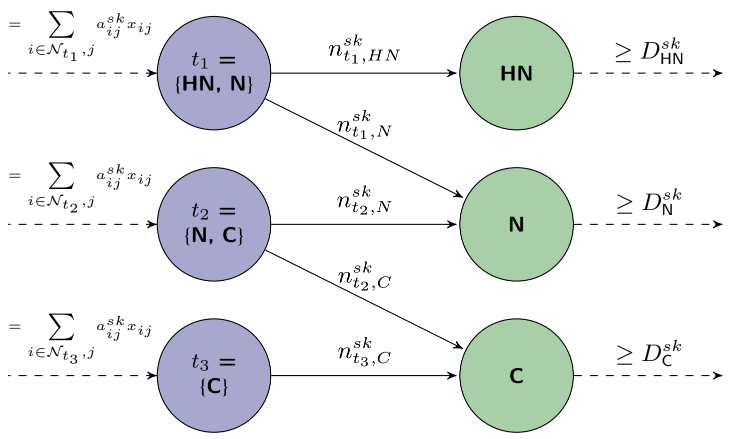
Proposition 1. *Assume that all x_{ij} take integral values. Then, any extreme solution $n_{t\sigma}^{sk}$ of the set of linear constraints above (**H2**, **H4**, **S1**) is also integral.*

Proof. From the integer programming theory, we know that all extreme solutions to a min-cost flow problem with integral inputs/outputs are integral. We show here that the allocation of skills can be described by a min-cost flow problem in a bipartite graph with integral inputs/outputs. The two sets of vertices of this bipartite graph are $\mathcal{T}$ and Σ. For a type t and a skill σ, (t, σ) is an arc of the graph if and only if $\sigma \in t$. The incoming flow at node t is $\sum_{i \in \mathcal{N}_\sigma, j} a_{ij}^{sk} x_{ij}$, and the flow exiting node σ must be greater or equal to D_σ^{sk}. Since these terms are integral, the proposition holds. An example of such a graph is given in Figure 2. $\square$

3.4. Integer Programming formulation

In this section, we summarize our IP for the NSP, based on the rotations previously defined. The complete model is given by the equations (1a)–(1l). Unless stated otherwise, the indices belong to the following sets: $i \in \mathcal{N}$, $k, l \in \{1 \ldots K\}$, $s \in \mathcal{S}$, $t \in \mathcal{T}$, $\sigma \in \Sigma$, and for each nurse i, $j \in \Omega_i$. The greek letters indicated between brackets (α, β, γ and δ) denote the dual variables associated with these constraints. Only the dual variables associated with constraints where x_{ij} variables appear are needed in the rotation-generation procedure, therefore, we only provide notations for those. It includes the aformentioned flow constraints from the rostering graph of Section 3.2, and the allocation constraints of Section 3.3.

Parameters and variables. All the parameters and variables used in the IP are described in Table 4. The complete list of parameters should also include the input data given in Table 1.



$= \sum_{i \in \mathcal{N}_{t_1}, j} a_{ij}^{sk} x_{ij}$
$t_1 =$ {HN, N}
$n_{t_1,HN}^{sk}$
HN
$\geq D_{\text{HN}}^{sk}$
$n_{t_1,N}^{sk}$
$= \sum_{i \in \mathcal{N}_{t_2}, j} a_{ij}^{sk} x_{ij}$
$t_2 =$ {N, C}
$n_{t_2,N}^{sk}$
N
$\geq D_{\text{N}}^{sk}$
$n_{t_2,C}^{sk}$
$= \sum_{i \in \mathcal{N}_{t_3}, j} a_{ij}^{sk} x_{ij}$
$t_3 =$ {C}
$n_{t_3,C}^{sk}$
C
$\geq D_{\text{C}}^{sk}$

Figure 2: Example of a graph for the skill allocation problem. Here, the set of skills is $\Sigma = \{$Head Nurse (HN), Nurse (N), Caretaker (C)$\}$, and the three possible types are $t_1 = \{$Head Nurse, Nurse$\}$, $t_2 = \{$Nurse, Caretaker$\}$, and $t_3 = \{$Caretaker$\}$. Dashed arcs show the flow constraints that must be satisfied (**H4** on the left, and **H2** on the right of the figure).

Parameters	
c_{ij}	sum of the soft penalties **S2**, **S4** and **S5** associated with rotation j of nurse i
c_3^{ikl}	soft penalty **S3** associated with the rest arc (W_{ik}, W_{il})
a_{ij}^{sk}	$= 1$ if nurse i works on day k, shift s in rotation j, 0 otherwise
a_{ij}^{k}	$= 1$ if nurse i works on day k in rotation j, 0 otherwise (i.e., $= \sum_s a_{ij}^{sk}$)
b_{ij}^{m}	$= 1$ if weekend m is worked in rotation j of nurse i, 0 otherwise (i.e., $= \max(a_{ij}^{7m-1}, a_{ij}^{7m})$)
f_{ij}^{-}, f_{ij}^{+}	first and last worked days of rotation j of nurse i

Variables		
x_{ij}	$\in \{0,1\}$	$= 1$ if and only if rotation j is part of the schedule of nurse i
r_{ik}	$\in \{0,1\}$	$= 1$ if and only if nurse i rests on day k and has already rested for at least CR_i^+ consecutive days before k (cost: c_3)
r_{ikl}	$\in \{0,1\}$	$= 1$ if and only if nurse i has a rest period from day k to $l-1$ including at most CR_i^+ consecutive days (cost: c_3^{ikl})
$w_i^+ (w_i^-)$	$\in \mathbb{N}$	number of days worked above (below) $L_i^+ (L_i^-)$ by nurse i
v_i	$\in \mathbb{N}$	number of weekends worked above B_i by nurse i
n_σ^{sk}	$\in \mathbb{N}$	number of nurses performing skill σ on shift (k,s)
$n_{t\sigma}^{sk}$	$\in \mathbb{N}$	number of nurses of type t performing skill σ on shift (k,s)
z_σ^{sk}	$\in \mathbb{N}$	number of missing nurses performing skill σ on shift (k,s)

Table 4: Parameters and variables of the IP (Formulation 1).

Objective and constraints. The objective function (1a) is the sum of 5 terms: the cost of the chosen rotations in terms of consecutive assignments and preferences (**S2**, **S4**, **S5**), the minimum and maximum consecutive rest days violations (**S3**), the total number of assignments violation (**S6**), the total number of worked weekends violation (**S7**), and the insufficient staff for optimal coverage (**S1**). Constraints (1b)–(1d) are the flow constraints of the rostering graph of each nurse $i \in \mathcal{N}$. Constraints (1e) and (1f) measure the distance between the number of worked days and the authorized number of assignments: the nonnegative variables w_i^+ and w_i^- respectively represent the number of missing days when the minimum number of worked days L_i^- is not reached and the number of worked days over the maximum allowed when the total number of days worked exceeds the maximum L_i^+. Constraints (1g) measure the number of worked weekends exceeding the maximum B_i (corresponding variable: $v_i \geq 0$). Constraints (1h)–(1j) ensure a valid allocation of the skills among nurses of a same type for each day and shift and have already been presented in Section 3.3.

Constraints (1k) and (1l) ensure the integrality and the nonnegativity of the decision variables. From constraint (1i), we know that the z_σ^{sk} are integer whenever the $n_{t\sigma}^{sk}$ are. Furthermore, from Proposition 1, we know that the integrality of the $n_{t\sigma}^{sk}$ is a consequence from that of the x_{ij}. Therefore, $x_{ij} \in \mathbb{N}$ and $z_\sigma^{sk}, n_{t\sigma}^{sk} \in \mathbb{R}$ is sufficient to ensure the integrality of all these variables.

$$\begin{aligned}\min \underbrace{\sum_{i,j} c_{ij} x_{ij}}_{\mathbf{S2},\mathbf{S4},\mathbf{S5}} + \underbrace{\sum_{i,k}\left(c_3 r_{ik} + \sum_{l} c_3^{ikl} r_{ikl}\right)}_{\mathbf{S3}} \\ + \underbrace{c_6 \sum_{i} (w_i^+ + w_i^-)}_{\mathbf{S6}} + \underbrace{c_7 \sum_{i} v_i}_{\mathbf{S7}} + \underbrace{c_1 \sum_{s,k,\sigma} z_\sigma^{sk}}_{\mathbf{S1}}\end{aligned} \tag{1a}$$

subject to:

$$[\mathbf{H1},\mathbf{H3}] \quad \sum_{l=k+1}^{\max(K+1,k+R_i)} r_{ikl} - \sum_{j:f_{ij}^+=k-1} x_{ij} = 0, \qquad \forall i, \forall k \quad [\alpha_{ik}^R] \tag{1b}$$

$$[\mathbf{H1},\mathbf{H3}] \quad r_{ik} - r_{i(k-1)} + \sum_{j:f_{ij}^-=k} x_{ij} - \sum_{l=\max(1,k-R_i)}^{k-1} r_{ilk} = 0, \qquad \forall i, \forall k \quad [\alpha_{ik}^W] \tag{1c}$$

$$[\mathbf{H1},\mathbf{H3}] \quad \sum_{l=1,K+1-R_i}^{K} r_{ilK} + r_{iK} + \sum_{j:f_{ij}^+=K} x_{ij} = 1, \qquad \forall i \quad [\alpha_{i(K+1)}] \tag{1d}$$

$$[\mathbf{S6}] \quad \sum_{j,k} a_{ij}^k x_{ij} + w_i^- \geq L_{\kappa_i}^-, \qquad \forall i \quad [\beta_i^-] \tag{1e}$$

$$[\mathbf{S6}] \quad \sum_{j,k} a_{ij}^k x_{ij} - w_i^+ \leq L_{\kappa_i}^+, \qquad \forall i \quad [\beta_i^+] \tag{1f}$$

$$[\mathbf{S7}] \quad \sum_{j,m} b_{ij}^m x_{ij} - v_i \leq B_{\kappa_i}, \qquad \forall i \quad [\gamma_i] \tag{1g}$$

$$[\mathbf{H2}] \quad \sum_{t\in\mathcal{T}_\sigma} n_{t\sigma}^{sk} \geq D_\sigma^{sk}, \qquad \forall s,k,\sigma \tag{1h}$$

$$[\mathbf{S1}] \quad \sum_{t\in\mathcal{T}_\sigma} n_{t\sigma}^{sk} + z_\sigma^{sk} \geq O_\sigma^{sk}, \qquad \forall s,k,\sigma \tag{1i}$$

$$[\mathbf{H4}] \quad \sum_{i\in\mathcal{N}_t,j} a_{ij}^{sk} x_{ij} - \sum_{\sigma\in\Sigma\mathcal{T}_t} n_{t\sigma}^{sk} = 0, \qquad \forall s,k,t \quad [\delta_t^{sk}] \tag{1j}$$

$$x_{ij} \in \mathbb{N}, z_\sigma^{sk}, n_{t\sigma}^{sk} \in \mathbb{R}, \qquad \forall i,j,s,k,t,\sigma \tag{1k}$$

$$r_{ikl}, r_{ik}, w_i^+, w_i^-, v_i \geq 0, \qquad \forall i,k,l \tag{1l}$$

4. Solution of the integer program by branch and price

A branch-and-price algorithm embeds a column generation within a classical branch-and-bound scheme to solve linear programs with integrality constraints. In this framework, every linear relaxation that occurs in the branching tree is solved by column generation and specific branching rules are designed. The reader looking for more details on both column generation and branch and price is referred to the textbook of

Desaulniers *et al.* [8]. In Section 4.1, we describe the overall column generation scheme that we implemented to solve the linear relaxation of Formulation (1a)–(1l). In Section 4.2, we describe our model for the pricing subproblem. In Section 4.3, we detail the branching rules implemented in the branch-and-price algorithm.

4.1. Description of the column generation procedure

Suppose that for each nurse $i \in \mathcal{N}$, a restricted number of rotations $\mathcal{R}_i \subseteq \Omega_i$ has already been generated. The restricted master problem (RMP) is equal to the IP of Formulation (1a)–(1l) where Ω_i is replaced by $\mathcal{R}_i$ for all nurses $i \in \mathcal{N}$. For the sake of simplicity, we assume that the linear relaxation of RMP, RMP^{LR}, is feasible – a feasible solution can always be obtained by adding artificial variables at prohibitive cost. Let $\boldsymbol{x}^{LR}$ be an optimal solution of RMP^{LR}. The subproblem described in the following sections is then solved to search for rotations of $\Omega_i \setminus \mathcal{R}_i$ with negative reduced costs. If at least one rotation of negative reduced cost is generated for at least one nurse i, it is added to the restricted formulation; if none is found, $\boldsymbol{x}^{LR}$ is then proved to be optimal for the linear relaxation of Formulation (1a)–(1l).

The resulting decomposition of the constraints corresponding to our IP is summarized on Table 5. The subproblems generate new rotations, while the master problem implements Formulation (1a)–(1l). The only constraint that appears on both ends of the decomposition is the single-assignment-per-day constraint **H1**: in the master problem, no pair of rotations with an assignment on the same day should be selected for the same nurse, and in the subproblem, no rotation with two assignments on the same day should be constructed.

Master Problem	Subproblem
H1 Single assignment per day	**H1** Single assigmnent per day
H2 Under-staffing	**H3** Shift type succession
H4 Missing required skill	**S2** Consecutive assignments
S1 Optimal coverage	**S4** Preferences
S3 Consecutive days off	**S5** Complete week-end
S6 Total assignments	
S7 Total working week-ends	

Table 5: Decomposition of the constraints in the master and sub-problem.

4.2. Pricing subproblem

The pricing subproblem for the generation of rotations with negative reduced costs is modeled as an SPPRC whose description is given below. The hard constraints of the subproblem are enforced by the structure of the network, whereas the master problem's dual costs and the penalties associated with the preference and complete-week-end soft constraints define the arcs costs, as described in Section 4.2.1. As for the consecutive assignments constraints they both correspond to the consumption of resources (consecutive assignments) along the path defining a rotation. Due to the presence of upper and lower bounds and to the possibility of violating these at the cost of a penalty, the soft constraints **S2** require a special treatment to be considered in an SPPRC. The corresponding modification we apply to the network are described in Section 4.2.2, and we assess the resulting complexity of the best existing labeling algorithms on the resulting network in Section 4.2.3.

4.2.1. Overview of the pricing problem network

We lay here the foundations of the pricing problem and present a basic version of the network of the SPPRC. The graph that we describe here is very similar to many others met in scheduling applications. For shift (k, s), a node W_{ks} is created and each path from source to sink that goes through W_{ks} involves working on (k, s). The vertices and arcs of the network are detailed in Table 6 and a small example is given in Figure 3

Vertices	*[source], [sink]*	One source node, one sink node.
	[shift nodes]	For each day k and shift s, one node W_{ks}.
Arcs	*[starting arcs]*	For each shift (k, s), one arc from source to W_{ks}.
	[successions]	For each day $k \geq 1$ and allowed shift succession $(s_1, s_2) \notin \bar{\mathcal{F}}$, one arc $\left(W_{(k-1)s_1}, W_{ks_2}\right)$.
	[ending arcs]	For each shift (k, s), one arc from W_{ks} to sink.

Table 6: Description of the vertices and arcs of the basic version of the pricing network.

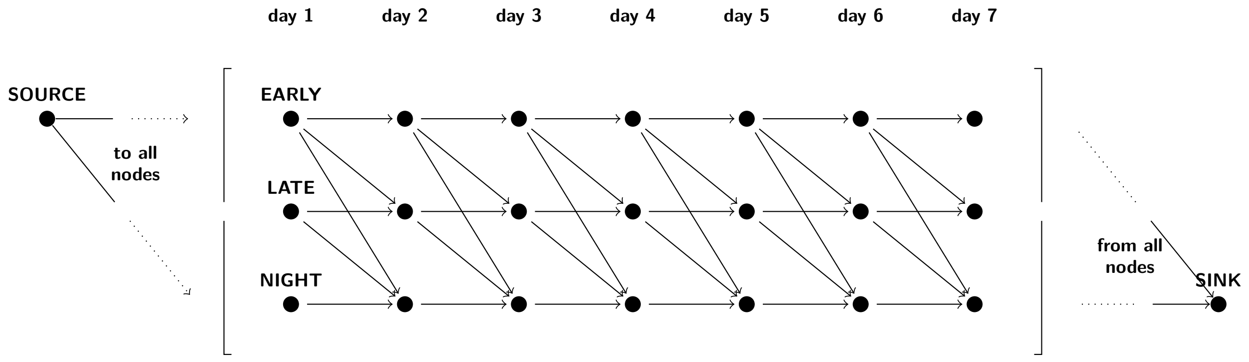


Figure 3: Example of a network of the subproblem where **S2** is not considered. Here, $K = 7$, $\mathcal{S} = \{\text{EARLY}, \text{LATE}, \text{NIGHT}\}$ and the forbidden successions are $\bar{\mathcal{F}} = \{(\text{LATE}, \text{EARLY}), (\text{NIGHT}, \text{EARLY}), (\text{NIGHT}, \text{LATE})\}$.

By construction of the network of Figure 3, any path from the source to the sink corresponds to a feasible rotation. To prove that an optimal solution of RMP^{LR}, $\boldsymbol{x}^{LR}$, is optimal for the linear relaxation of Formulation (1a)–(1l), it must be shown that no rotation, wether involved in RMP^{LR} or not, is of negative reduced cost at $\boldsymbol{x}^{LR}$. The idea is to compute the reduced cost of a rotation by simply adding up the costs of its arcs in the network of the subproblem (Figure 3). A feasible rotation of lowest reduced cost is then a shortest path from the sink to the source.

Proposition 2. *Let $j \in \Omega_i$ be a feasible rotation for nurse $i \in \mathcal{N}$, $((k_b, s_{k_b}), (k_b + 1, s_{k_b+1}), \ldots, (k_e, s_e))$ be the sequence of worked shifts in the rotation, and N^j_{we} be the number of week-ends with at least one assignment in the rotation. The reduced cost of rotation j is*

$$\bar{c}_{ij} = \underbrace{c_{ij}}_{\textit{S2,S4,S5}} + \underbrace{\alpha^W_{ik_b} + \alpha^R_{ik_e}}_{(1\text{b})-(1\text{d})} + \underbrace{\left(\beta^+_i + \beta^-_i\right)(k_e - k_b + 1)}_{(1\text{e})-(1\text{f})} + \underbrace{\gamma_i N^j_{we}}_{(1\text{g})} + \underbrace{\sum_{k=k_b}^{k_e} \delta^{s_k k}_i}_{(1\text{j})}, \tag{2}$$

where the variables α, β, γ and δ are dual variables of Formulation (1a)–(1l).

Proof. The variable that corresponds to the rotation $j \in \Omega_i$ in Formulation (1a)–(1l) is x_{ij}. It is then sufficient to observe that the first term of (2) is the cost of the rotation and the remaining terms are the sums of the dual variables associated with the constraints of (1a)–(1l) where x_{ij} appears weighted by the coefficients of x_{ij} in these constraints. □

Among the terms that define $\bar{c}_{ij}$ (Equation (2)), the soft penalties **S4**, **S5** and all dual costs can be added directly as arc costs to the network presented before. The costs associated with the nonrespect of the nurse's preferences **S4** result in the addition of a penalty c_4 on all arcs whose endpoint is an undesired shift. The penalty associated to the complete week-end constraint **S5** is paid only if the rotation starts on a Sunday or ends on a Saturday, hence a cost c_5 is added to every arc from the source to Sunday shifts and from Saturday shifts to the sink. Now, for the dual costs, $(\beta_i^+ + \beta_i^-)$ must be paid for each assignment, and $\delta_i^{s_k k}$ is paid for an assignment to shift (k, s), so we add $(\beta_i^+ + \beta_i^-) + \delta_i^{s_k k}$ to the cost of the arc with endpoint W_{ks} for all (k, s). Then, $\alpha_{ik_b}^W$ and $\alpha_{ik_e}^R$ correspond to the starting and ending shift of the rotation so they are respectively added to every outgoing arc from the source and every incoming arc to the sink. Finally, to penalize the number of week-ends with at least one assignment, we add γ_i to every incoming arc to a Saturday shift and to every arc from the source to any week-end shift.

Contrary to the above costs, the penalties associated with the constraints on consecutive assignments **S2** cannot be attributed to one specific shift. Instead, they require to count the total number of shifts in the rotation and the number of consecutive assignments to the same type of shift, and then penalize the possible violations of lower and upper bounds on these values.

4.2.2. Modifications of the pricing network for consecutive assignments constraints

The classical method for "counting" a value over a path (e.g., the number of days) is to add a resource that measures this value on the arcs; the aggregate value is then the sum of the values of the resource on the arcs of this path [14]. The standard version of the SPPRC deals with acyclic digraphs where the resources are weighted with positive integer values, and the aggregate value must remain below a given upper bound. Lower bounds are usually associated with the start of time windows in delivery problems, but the vehicles are allowed to wait for the beginning of the time window, which does not correspond to any reality in our application. Instead, we need to deal with both lower and upper bounded soft constraints.

Upper and lower bounded soft constraints have already been considered in an SPPRC to solve variants of the vehicle routing problem where it is forbidden to "wait" at a node. Dumas et al. [9] include soft time windows to schedule deliveries in a network where the paths are already fixed. Braekers and Janssens [4] modify a labeling algorithm to be able to apply dominance rules with soft time windows, and they apply their algorithm to small toy instances. Qurashi et al. [19] solve the same variant of the SPPRC using several heuristics without any guarantee of optimality and the same authors solve an IP to find an exact solution of the problem [20]. The limit is that they fail in solving problems routing problems with more than 7 vehicles and 25 customers.

Overall, the algorithmic impact of considering upper and lower bounded soft constraints is twofold. First, the complexity for the standard SPPRC is $\mathcal{O}\left(A \prod_{r\in\mathcal{R}} M_r\right)$, where A is the number of arcs in the network, $\mathcal{R}$ is the set of resources, and M_r is the upper bound on resource $r \in \mathcal{R}$ [12]. This result relies on the infeasibility of paths with resource values larger than the upper bound in the standard SPPRC. In our case, though, soft constraints mean that the values of the resources can take higher values than the upper bounds. Second, the labeling algorithms that efficiently solve the standard SPPRC rely on dominance rule to converge quickly. Since the consecutive assignments are both upper and lower bounded, one can never conclude on the dominance of one path towards another based on the value of a resource: a smaller number of assignments is better with respect to the maximum, but worse with respect to the minimum, and the inverse is true for a larger number of assignments.

For the above-mentioned reasons, we had rather modify the pricing network to reduce subproblems to a standard SPPRC in a larger graph.

We describe here the modifications of the network that allow us to handle the penalties on the min/max of consecutive days worked on the same shift (addition of network layers and arcs) and the min/max of consecutive days worked (enumeration of short rotations and addition of a resource). The nodes and arcs of the resulting network are described in Table 7. We then assess the complexity of the solution algorithm for the subproblems in Section 4.2.3.

Remark. *The above-mentioned difficulties with lower and upper bounded soft constraints also influenced our choice not to generate full rosters but only rotations: each of the consecutive worked days/shifts/rest days soft constraints require the addition of two resources (or additional network layers). In the rotation-based model, rest days are set in the* RMP*, thus sparing the additional complexity in the solution of the subproblem.*

Consecutive assignments to the same shift. The penalty for N consecutive assignments to a given shift s is $c_{2b}\left(\mathrm{CS}_s^- - N\right)$ if $N < \mathrm{CS}_s^-$, 0 if $\mathrm{CS}_s^- \leq N \leq \mathrm{CS}_s^+$, and $c_{2b}\left(N - \mathrm{CS}_s^+\right)$ if $N > \mathrm{CS}_s^+$. Instead of adding one resource to penalize N when it is below CS_s^- and another when it is above CS_s^+, we model this constraint by adding network layers and arcs as follows. We duplicate each work node $(W_{ks}, \forall(k,s))$ CS_s^+ times; the duplicates are denoted as W_{ks}^n, $n = 1, \ldots, \mathrm{CS}_s^+$. A path going through node W_{ks}^n will then correspond to a rotation in which (k,s) is assigned and is either the n-th consecutive shift s, or *at least* the n-th one when $n = \mathrm{CS}_s^+$.

For each type of shift s, the work nodes form a block $\mathcal{W}_s$. We draw an example focussing on a specific block on Figure 4. The arcs entering the block $\mathcal{W}_s$ denote the beginning of a sequence of assignments to the shift s, and the arcs exiting from the block denote the end of this sequence. The first $\mathrm{CS}_s^+ - 1$ assignments to the shift s are associated with the plain arcs $(W_{ks}^n, W_{(k+1)s}^{n+1})$, and the subsequent assignments correspond to the dotted horizontal arcs $(W_{ks}^{\mathrm{CS}_s^+}, W_{(k+1)s}^{\mathrm{CS}_s^+})$. A cost c_{2b} is then added to the horizontal arcs because they are used only when more than CS_s^+ consecutive assignments of the shift have occurred, whereas the costs of the plain and diagonal arcs takes no penalty due to **S2**. Notice that the arcs exiting from the block only leave from vertices $W_{ks}^{\mathrm{CS}_s^+}, k \in \{1, \ldots, K\}$. To end a sequence of assignments to s before CS_s^+ consecutive assignments, the path must then borrow a vertical dashed arc, which allows us to model the penalty incurred if the number of consecutive shifts is too small $(\leq \mathrm{CS}_s^-)$.

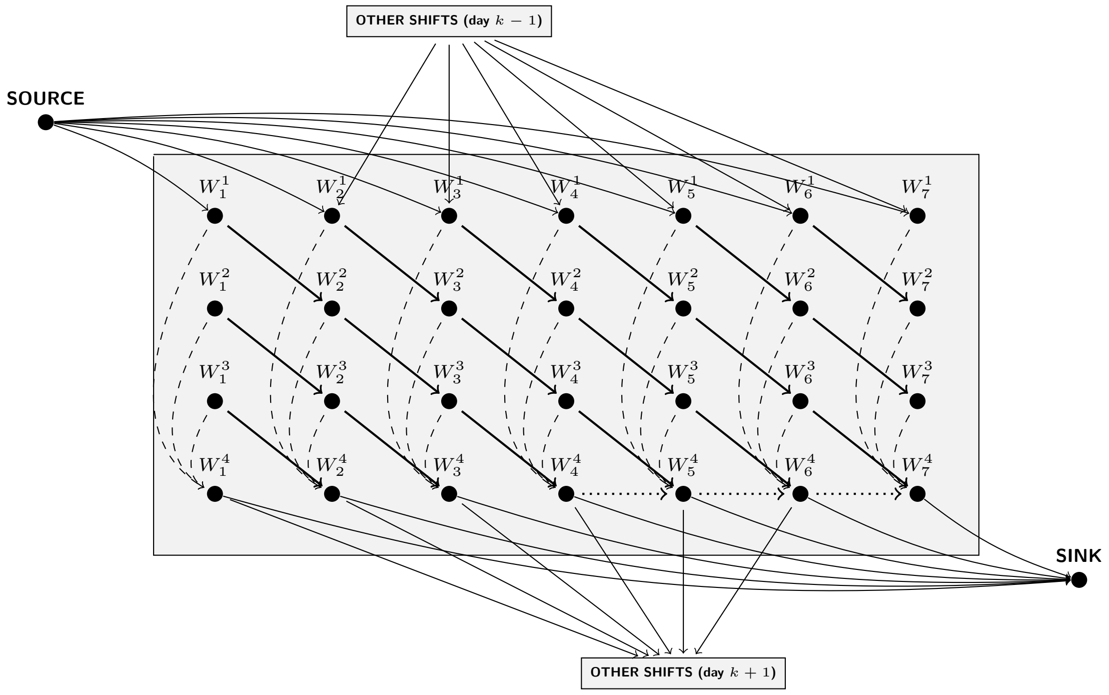


Figure 4: Example of a network of the subproblem of nurse i for a single shift s where **S2** is considered. Parameters: $K = 7$, $\mathrm{CS}_s^+ = 4$, and $\mathrm{CD}_i^- = 2$.

Consecutive worked days. For nurse $i \in \mathcal{N}$, the consecutive-worked-days penalty associated with a rotation of length L is $c_{2a}\left(\mathrm{CD}_i^- - L\right)$ if $L < \mathrm{CD}_i^-$, 0 if $\mathrm{CD}_i^- \leq L \leq \mathrm{CD}_i^+$, and $c_{2a}\left(L - \mathrm{CD}_i^+\right)$ if $L > \mathrm{CD}_i^+$. Given that CD_i^- is small in practice (2 or 3), we handle the lower bound by enumerating the rotations of length $L < \mathrm{CD}_i^-$ by dynamic programming. The reduced cost of these *short* rotations $(L < \mathrm{CD}_i^-)$ is thus computed as a preprocessing, and we modify the pricing network so that the arcs outgoing from the source correspond to a sequence of CD_i^- consecutive worked days (called *starting arcs* in Table 7). No path in the network represents a short rotation anymore and, therefore, there is no need to check the lower bound CD_i^- in the network. We study the resulting complexity of this approach in Section 4.2.3.

To handle the upper bound, we then add a resource that counts (a lower bound on) the length of the rotation. The value of this resource is CD_i^- on the starting arcs, 1 on the arcs that represent a worked day, and 0 on the others (i.e., the vertical dashed arcs of Figure 4 and the arcs to the sink). If the aggregate value of this resource at the end of the path exceeds CD_i^+, the corresponding penalty is added to the cost of the rotation. The vertices and the arcs of the subnetwork associated with one specific shift s, as well as the cost associated with the consecutive assignment constraints are summarized in Table 7.

Vertices	*[artificial nodes]*	One source, one sink.
	[shift nodes]	For each day k and all $1 \leq n \leq \mathrm{CS}_s^+$, a node W_{ks}^n.
	[penalty nodes]	For $n = \mathrm{CD}_i^+, \dots, K$, a node CD^n.
Arcs	*[starting arcs]*	For each possible sequence of exactly CD_i^- days ending with exactly n consecutive occurrences of shift s, for all days $k \geq \mathrm{CD}_i^-$, an arc from the source to W_{ks}^n. This arc corresponds to a rotation beginning on day $k-(\mathrm{CD}_i^- - 1)$ that reaches day k with an ongoing capital of n consecutive days worked on shift s. *Cost:* Consecutive cost attributable to shifts prior to the n consecutive ones + all other costs attributable to shifts in the CD_i^- days.
	[repeat s]	For all $k \geq \mathrm{CD}_i^- + 1$, for all $2 \leq n \leq \mathrm{CS}_s^+$, one arc $(W_{(k-1)s}^{n-1}, W_{ks}^n)$ corresponds to the nurse working on shifts (k, s) when already working on $(k-1, s)$.
	[exceed s]	For all $k \geq \mathrm{CD}_i^- + 1$, one arc $(W_{(k-1)s}^{\mathrm{CS}_s^+}, W_{ks}^{\mathrm{CS}_s^+})$. These are similar to the previous ones, but only apply when the nurse has already worked at least CS_s^+ consecutive days on shift s. *Cost:* c_{2b} (the nurse has already exceeded CS_s^+ consecutive shifts s).
	[end s]	For all $k \geq \mathrm{CD}_i^-$, for all $0 \leq n \leq \mathrm{CS}_s^+ - 1$, one arc $(W_{ks}^n, W_{ks}^{\mathrm{CS}_s^+})$. These arcs contain the cost of working less than CS_s^- consecutive days on shift s (or 0 if $\mathrm{CS}_s^- \leq n \leq \mathrm{CS}_s^+ - 1$). *Cost:* $c_{2b} \times \max\{0, n - \mathrm{CS}_s^-\}$.
	[change shift]	For all $k \geq \mathrm{CD}_i^- + 1$, and all shifts s' such that (s, s') is not a forbidden succession, one arc $(W_{(k-1)s}^{\mathrm{CS}_s^+}, W_{ks'}^1)$.
	[ending arcs]	For all $k \geq \mathrm{CD}_i^-$, one arc from $W_{ks}^{\mathrm{CS}_s^+}$ to the sink.

Table 7: Description of the vertices and arcs of a single-shift subnetwork of the pricing problem for shift $s \in \mathcal{S}$ and nurse $i \in \mathcal{N}$. Arc costs correspond only to those attributable to consecutive assignments constraints, and they are indicated only when they are nonzero.

4.2.3. Assessment of the complexity of the SPPRC in the network of the subproblem

For an assessment of the complexity of the subproblem, we first need to count the number of arcs in the pricing network. The total number of starting arcs is given in Lemma 1.

Lemma 1. *The number of starting arcs ξ_i in the network of the subproblem of nurse i satisfies*

$$\xi_i \leq |\mathcal{S}| \left(K - \mathrm{CD}_i^- + 1\right) \mathrm{CD}_i^- \tag{3}$$

Proof. The starting arcs all correspond to a sequence of CD_i^- assignments so they are all equivalent with respect to the resource on consecutive assignments. As a consequence, it is only necessary to keep the starting with minimum cost among those that income to a given work node W^n_{ks}. What is more, for a given shift s, no more than CD^-_i consecutive assignments to s can be done in the sequence represented by a starting arc, so there is no starting arc incoming to a work node W^n_{ks} for $n > \mathrm{CD}^-_i$. For each shift $s \in \mathcal{S}$, and each day $k \in \left\{\mathrm{CD}^-_i, \ldots, K\right\}$, we thus have at most 1 starting arc incoming to W^n_{ks} for $n \leq \mathrm{CD}^-_i$ and none to the other vertices of the network. The enumeration of the vertices with at most one incoming arc give the result. □

Note that Equation (3) is indeed an inequality (and not an equality in general) because some of the starting arcs enumerated above may not exist. For example, if a shift **EARLY** can only be performed after another shift **EARLY**, then no short rotation of 2 days can end with a single **EARLY** task, and $W^1_{k,\textsf{EARLY}}$ is never reached by a starting arc. Therefore, Equation (3) cannot be written as an equality.

Lemma 2. *The number of arcs in the pricing network of nurse i satisfies*

$$|A_i| = \mathcal{O}\left(K \times |\mathcal{S}| \times (|\mathcal{S}| + \mathrm{CD}^-_i + 2\max_{s\in\mathcal{S}}\{\mathrm{CS}^+_s\})\right) \tag{4}$$

Proof. The total number of arcs in the pricing network is exactly

$$\begin{aligned}
|A_i| = & \xi_i + |\mathcal{S}|\,(K - \mathrm{CD}^-_i) && \text{(5a)}\\
& + \sum_{s\in\mathcal{S}} (K - \mathrm{CD}^-_i) \times (\mathrm{CS}^+_s - 1) && \text{(5b)}\\
& + \sum_{s\in\mathcal{S}} (K - \mathrm{CS}^+_s) && \text{(5c)}\\
& + \sum_{s\in\mathcal{S}} (K - \mathrm{CD}^-_i) \times (\mathrm{CS}^+_s - 1) && \text{(5d)}\\
& + (|\mathcal{F}| - |\mathcal{S}|) \times (K - \mathrm{CD}^-_i - 1), && \text{(5e)}
\end{aligned}$$

where $\mathcal{F}$ is the set of allowed shift successions (i.e. the complement of $\overline{\mathcal{F}}$ in $\mathcal{S}^2$). The term (5a) stands for starting arcs and arcs incoming to the sink, (5b) counts the working arcs, (5c) is the number of horizontal dotted arcs on Figure 4, (5d) is the number of vertical dashed arcs on Figure 4, and (5e) is the number of arcs between different shift blocks. We get the result of the lemma by keeping only the positive terms and observing that there is at most $|\mathcal{S}| \times (|\mathcal{S}| + 1)$ shift successions allowed. □

The complexity of the dynamic programming algorithms for an SPPRC in an acyclic network with nonnegative integer weights is $\mathcal{O}\left(A \prod_{r\in\mathcal{R}} M_r\right)$, where A is the number of arcs in the network, $\mathcal{R}$ is the set of resources, and M_r is the upper bound on resource $r \in \mathcal{R}$ [12]. We can thus infer the complexity of dynamic programming for the column generation subproblem.

Theorem 1. *Consider a nurse $i \in \mathcal{N}$. Then a minimum reduced cost rotation for i can be computed by a dynamic programming algorithm with a complexity in*

$$\mathcal{O}\left(K^2 \times |\mathcal{S}| \times \left(|\mathcal{S}| + \mathrm{CD}^-_i + 2\max_{s\in\mathcal{S}}\{\mathrm{CS}^+_s\}\right)\right) \tag{6}$$

Proof. It is sufficient to state that a rotation with minimum reduced cost can be computed by solving an SPPRC in an acyclic network with nonnegative integer weights including $|A_i|$ arcs and one resource. The upper bound on the resource is indeed K (and not CD^+_i), because the corresponding constraint is soft. Therefore, a rotation can include at most K worked days. □

Remark. *For a given application, the values of $|\mathcal{S}|$, CD^-_i and CS^+_s will not vary much, so an asymptotical study of the complexity should only consider K as variable. What is more, the penalty for working more than CD^+_i consecutive days is among the highest penalties in the instances of our benchmark, so the rotations with* *a large number of extra consecutive assignments will generally be dominated in the dynamic programming algorithm. Hence, the complexity is most likely to be an asymptotical*

$$\mathcal{O}\left(K \times \mathrm{CD}_i^+ \times |\mathcal{S}| \times \left(|\mathcal{S}| + \mathrm{CD}_i^- + 2\max_{s\in\mathcal{S}}\{\mathrm{CS}_s^+\}\right)\right).$$

Typical contractual values are $|\mathcal{S}| = 4, \mathrm{CD}_i^- = 3, \mathrm{CD}_i^+ = 5$ *and* $\mathrm{CS}_s^+ = 3$*, so the execution of the subproblems is in practice in* $\mathcal{O}(310K)$.

4.3. Branching rules

In this section, we describe the two branching rules implemented in our branch-and-price algorithm. We then detail how the branching rule is chosen at each node of the branch-and-bound tree. Finally, we present a heuristic used for the determination of feasible (integral) solutions. In the descriptions of the branching rules, we consider a specific node N of the branch-and-bound tree and denote $\boldsymbol{x}_N^{LR}$ the optimal solution of the linear relaxation at node N.

4.3.1. Branching on days

For every nurse i and day k, denote as

$$g_{ik} = \sum_{(l_1,l_2):\leq k<l_2} r_{il_1l_2} + r_{ik}$$

the value indicating whether the nurse i is resting on day k ($g_{ik} = 1$), or not ($g_{ik} = 0$). Assume that $\boldsymbol{x}_N^{LR}$ is such that the value of g_{ik} is fractional for some nurse i and day k. We create two branches from node N to ensure that g_{ik} is integer. In the first one (the *work branch*), nurse i is compelled to work on day k, and in the other one (the *rest branch*), he/she is compelled to rest. The corresponding constraints added to the master problem are, respectively,

$$g_{ik} = 0 \tag{7}$$

and

$$g_{ik} = 1. \tag{8}$$

In the rest branch, no more rotation should be generated that makes nurse i work on day k; this is easily handled by removing all arcs concerning day k in the subproblem of nurse i, and it even accelerates the solution of the subproblem since the network becomes smaller. In the work branch, the subproblem remains unchanged.

Remark. *The rotation-based model is particularly well-fitted to this branching rule. In the rest branch, one only discards* rotations *that include day k, but all other rotations remain in the problem. That is, none of the efforts made in the previous column-generation iterations that aimed at generating good patterns for the other days are wasted (e.g., rotations that cover the end of the schedule if k is in the beginning of the horizon).*

4.3.2. Branching on shifts

The value of g_{ik} can be integer for all nurses i and days k while still having fractional variables in $\boldsymbol{x}_N^{LR}$. For an exhaustive enumeration of the feasible schedules, we thus implement a branching rule, similar to the previous one, that affects working shifts. For all nurses i, days k and shifts s, denote as $\Omega_{iks} \subseteq \Omega_i$ the subset of the rotations for nurse i for which the nurse works on shift (k,s), and as $h_{iks} = \sum_{j\in\Omega_{iks}} x_{ij}$ the value indicating whether nurse i works on shift (k,s) ($h_{iks} = 1$) or not ($h_{iks} = 0$).

If nurse i works partially on shift (k,s), i.e., if h_{iks} is fractional, we create one branch where he/she works on (k,s), and another where he/she does not. The corresponding constraints added to the master problem are, respectively,

$$h_{iks} = 0 \tag{9}$$

and

$$h_{iks} = 1. \tag{10}$$

This branching rule is very similar to the previous one: the work shifts play the role of the rest shifts in the previous rule. The consequences on the master and subproblems are very similar to those described in the previous section.

4.3.3. Choice of the branching strategy

The nodes of the branch-and-price tree are explored in the following order: if children are created, explore one of them, otherwise, go back to the highest non-explored node of the tree. The sequence of exploration that starts from the highest nonexplored node and ends at a leaf (never going back to a higher node) is called a *dive.* At each node of the tree, the choice of the branching rule is done as follows:

- Priority is always given to branching on days over branching on shifts.
- When branching on days, we choose the most balanced fractional value. That is, the pair nurse-day (i, k) that satisfies
$$(i,k) = \operatorname*{arg\,min}_{(i,k)\in\mathcal{N}\times\{1\ldots K\}} f_{ik}, \tag{11}$$
where $f_{ik} = \|g_{ik} - 0.5\|$ if k is a week day, and $f_{ik} = \|g_{ik} - 0.5\| - 0.1$ if k is a weekend day. This corrective term gives a slight advantage to weekend days because they are involved in more soft constraints and therefore have more influence on the objective value.
- When branching on shifts, the same selection method is used (replace f_{ik} by $f_{iks} = \|h_{iks} - 0.5\|$ if k is a weekday and $h_{iks} = \|g_{iks} - 0.5\| - 0.1$ if k is a weekend day).
- The order in which the two children branching nodes are inserted is random.

4.3.4. Primal heuristic based on variable fixing to obtain feasible solutions

The solutions of the linear relaxations solved at the nodes of the branching tree are often fractional, so it is important to regularly run heuristics to determine integer solutions that improve the upper bound. Such heuristics are usually called *primal heuristics.* Our heuristic performs the following steps:

(i) For some threshold $C \in \,]0,1[$, for all $i \in \mathcal{N}$ choose a set of rotations Θ_i that do not overlap (i.e., contain no assignment on the same day) and are all separated by at least one rest day, such that
$$\sum_{i\in\mathcal{N}} \sum_{j\in\Theta_i^{ks}} (1 - x_{ij}^{LR}) \leq C, \quad \text{for all shifts}(k,s), \tag{12}$$
where $\Theta_i^{ks} \subseteq \Theta_i$ is the set of rotations in Θ_i that involve working on shift (k, s) (Θ_i^{ks} is either empty, or a singleton).

(ii) For all $i \in \mathcal{N}$ and all $j \in \Theta_i$, fix $x_{ij} = 1$.

(iii) Solve the problem with fixed variables (using column generation). If it is infeasible, the heuristic failed to find a solution. If it is feasible and the optimal solution is integer, return this solution. If it is feasible and the optimal solution is fractional, go to step (i).

The condition (12) is useful to fix only a small number of rotations that involve working on the same shift (k, s) and thus reduce the risk that the problem becomes infeasible.

The primal heuristic is run in the following two cases: (1) after the initial solution of the root node and (2) from the highest nonexplored node after each 2^q-th dive (q integer).

5. An adaptive large neighborhood search for large instances

Although the branch-and-price procedure presented in the previous sections allows the optimal solution of small problems, as such it is not best-suited for large instances. When the number of integer variables is too high, the size of the branching tree explodes and evaluating every node turns out to be extremely time-consuming. Therefore, we embed this branch and price in an adaptive LNS (ALNS) procedure based on the solution of smaller IPs, or equivalently, on the successive fixing and release of some of the variables (see [17] for a recent review on LNS).

An LNS algorithm is an iterative process that destroys a part of the current solution at each iteration and reconstructs it in the hope for an improvement. In the destruction phase, a subset of the variables are freed, while the rest is fixed to its current value; the choice of the fixed/free variables defines a neighborhood. In our implementation, the repair phase uses the branch-and-price method presented in the previous sections at the sole exception that only a subset of the variables are free to change value, the others being fixed. Typically, for the NSP, one can destroy the schedule of a subset of the nurses (freeing the corresponding rotations). Then, without changing the other nurses' schedules, the subset of destroyed schedules can be optimized by branch and price. This results in the solution of much smaller problems, involving a subset of the variables only (less nurses, less days, etc.), rather than the whole problem. The algorithm is therefore based on the repetition of the following two steps, until a stopping criterion is reached (time, number of iterations, number of iterations without improvement, ...):

1. *Destroy:* Use a destruction operator to determine a set (called FREE) of variables to free and fix the others (set FIXED) at their value in the current solution;
2. *Repair:* Solve the NSP by branch and price where all variables in FIXED are fixed to their value in the current solution. If the solution is improved, store it as the new current solution.

Using branch and price as the repair function is a double-edged sword: on the one hand, we benefit from the guarantees of an exact method, well suited for smaller problems, but on the other hand, the destruction operator must be compatible with the column-generation procedure. In practice, if the generation of new columns is made impossible by the structure of the destruction operator, the method is obviously doomed to failure. Interestingly, our model based on rotations yields a wider range of choices of neighbourhoods than a classical roster-based modelling.

In Section 5.1, we propose several different destruction operators (i.e., different neighborhoods) that adapt to the generation of rotations. What is more, we take advantage of the definition of several destruction operators by choosing the operator randomly at each iteration with probabilities that depend on their previous successes/failures. The corresponding roulette wheel procedure is described in Section 5.2. In Section 5.3, we describe a rolling-horizon procedure that we use to determine the initial solution fed to the ALNS.

5.1. Destruction operators

For the process of destructing the current solution, we propose two main strategies: destroy the schedule of a small set of nurses, or destroy the schedules of a larger set of nurses over a restricted time period. For both strategies, the nurses whose schedules are partially or completeley destroyed are called *free nurses*, and the others are called *fixed nurses*. In terms of implementation, if the schedule of a free nurse is completely destroyed, the variables associated with the rotations of the free nurse, and *all* the allocation and covering variables are set free. In contrast, the variables associated with the rotations of the fixed nurses are fixed to their current value.

Remark (Allocation variables)**.** *One should note the following: the aforementionned problem is* not *equivalent to solving a smaller problem for a restricted nursing service because allocation variables remain free and they concern* all *nurses. Therefore, there is a certain degree of flexibility in the allocation that one assigns to the free nurses, and two different re-optimized schedules for the free nurses may in practice satisfy a different partial demands. This difference is compensated by a change in the allocation of the fixed nurses, because only their rotations are fixed, but not their skill allocations.*

Thanks to the rotation-based model, the partial destruction of schedules can also be handled easily, because the rotations are much shorter than complete schedules. Given a starting and an ending date $k_1 < k_2$, the partial schedules from k_1 to k_2 are destroyed by freeing every rotation starting on any day k satisfying $k_1 \leq k \leq k_2$, as well as the corresponding allocation and covering variables, and all workload-related variables (resp., n, z, v, and w). To repair the solution, we adapt the branch-and-price algorithm by generating rotations that can start only between days k_1 and k_2. In practice, we modify the subproblem's network by deleting the starting arcs corresponding to rotations that start before k_1 or after k_2.

For a unified presentation of the destructors, we denote as N_W the number of schedule weeks destroyed per free nurse, and N_N the number of free nurses. Preliminary tests showed that there is no benefit in partial destruction of only one week in the schedule: with the rest of the schedules fixed, the problem is too constrained to make any improvement during the repair step. As a consequence, for a simpler implementation, we pick N_W in $\{2, 4, 8\}$ for an eight-weeks horizon and in $\{2, 4\}$ for a four-weeks horizon. The number of nurses is then chosen to get a constant total number of destroyed schedule weeks ($N_N \times N_W$). The product $N_N \times N_W$ is a parameter of the ALNS, whose influence is studied in Section 6.2.3.

As for the choice of the free nurses, we propose the following destruction strategies that depend on the number of free nurses, N_N.

i. `D_Random(`N_N`)`: N_N nurses are randomly selected among all the nurses;
ii. `D_Type(`N_N`,`T`)`: N_N nurses are randomly selected among those of the nurses with type T;
iii. `D_Contract(`N_N`,`C`)`: N_N schedules are randomly selected among the nurses that share the same contract C. The contract of a nurse i is defined by the values of $\left(L_i^-, L_i^+, \mathrm{CD}_i^-, \mathrm{CD}_i^+, \mathrm{CR}_i^-, \mathrm{CR}_i^+, B_i, \beta_i\right)$.

5.2. *Choice of the destruction operator: roulette-wheel procedure*

To select the destruction strategy of the nurses, and the number of weeks destroyed per nurse, we call the roulette-wheel procedure introduced in [18] twice at each iteration of the ALNS. To choose between `D_Random`, `D_Type` and `D_Contract` for instance, each destruction operator d is assigned a value π_d that starts at 5. Every time the operator d is selected and yields an improvement, π_d is incremented by 1. At the beginning of each ALNS iteration, the destruction operator is selected randomly, where each operator d having a probability $\pi_d / \left(\sum_{d'} \pi_{d'}\right)$ of being selected. The number of weeks N_W is then selected likewise, and so are the type T and the contract C when `D_Type` or `D_Contract` are selected.

In contrast, the sampling probability of the free nurses is not adjusted according to a roulette wheel procedure inside a given distribution operator (`D_Random/D_Type/D_Contract`). Nevertheless, the probability distribution for the nurses random selection is not uniform and is adjusted dynamically. A bias is added to increase the likelyhood to draw schedules where the number of assignments is not within a small margin from the average number of assignments per nurse. The motivation is to increase the possibility of obtaining a better workload repartition and decreasing the under/over workload and worked week-ends penalties.

5.3. *Improving the initialization: rolling-horizon algorithm*

The initial solution of the ALNS is found by running a rolling-horizon method over the planning horizon. Rolling-horizon methods come from the control theory area (Model Predictive Control) and are originally meant to solve problems where future data is uncertain (wether unknown, noisy or depending on external forces) [21]. In our implementation of this method, the planning horizon is chronologically partitioned into three time windows: *past*, *control horizon* (present and near future) and *prediction horizon* (further future) (see Figure 5). The variables that refer to past days are fixed, those of the control horizon are set to be integer, and those of the prediction horizon are relaxed, i.e., they are allowed to be fractional. The control horizon is the time period that we are *actually* scheduling, whereas the prediction horizon gives us an estimation of the impact and influence of the decisions taken for the control horizon on the future. After this problem has been solved, the windows are shifted towards the future by a chosen step called *sampling horizon*, until the algorithm reaches the end of the complete planning horizon.

The same kind of idea has been successfully implemented to tackle large aircrew pairing problems in [22] and led to substantial improvements. In the presence of uncertainty, rolling horizon performs best when the dynamics of the problem are slow. Here, the demand does not change between each step of the process

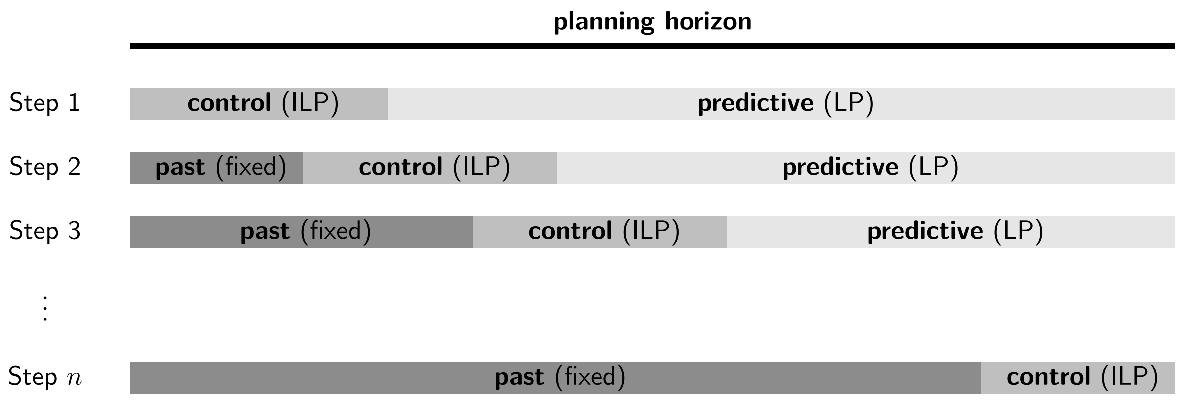


Figure 5: Rolling-horizon procedure. Variables corresponding to rotations starting in the past, control, and predictive horizon are respectively fixed, integer, and relaxed (i.e., may be fractionnal).

and we can say that there is no external dynamics, hence the algorithm should perform efficiently. As a matter of facts, the only dynamics generated is that of the influence of the history (the last days of the past horizon) over the beginning of the control horizon.

6. Experimentations

6.1. Instances and benchmark

Instances. We test our algorithm on the instances of the INRC–II competition[2]. The size of the service ranges from 30 to 120 nurses that can perform up to 4 different skills. The length of the planing horizon is either 4 or 8 weeks and each day is divided in up to 4 shifts. The data of each instance is not contained in a single file, but as follows: one file describes the available staff (number of nurses $|\mathcal{N}|$, contracts, etc.), another one contains the working status of the nurses at the begining of the horizon (history), and M files that each contains the demand for one of the weeks (recall that M is the number of weeks in the horizon). For each size of staff and horizon length, a single staff file, three different history files and 10 different week-demand files are available. This allows a huge number of combinations and the potential generation of thousands of instances. Each instance is named by the same pattern that completely describes the files defining it: n[$|\mathcal{N}|$]w[M]_*[history]*_*[demand files]*. As an example, the instance n030w4_1_6-2-9-1 describes a 30-nurses instance, over a 4-weeks planing horizon, with history file number 1, and week-demand files 6, 2, 9, and 1.

Benchmark. We consider a benchmark of 40 instances: for each staff size

$$|\mathcal{N}| \in \{30, 35, 40, 50, 60, 70, 80, 100, 110, 120\},$$

we consider two instances of four weeks, and two instances of eight weeks. More precisely, we use a subset of the instances that were created for the evaluation of the competing teams of the INRC–II competition (see [7]). All the instances are listed in Table 8 alongside with the best results found by our algorithm. Under the label “instance” are the names of the instances, “LB*” and “UB*” respectively denote the best lower and upper bound that we found, and “Gap” is the corresponding integrality gap, i.e., the value of $(\text{UB}^* - \text{LB}^*)/\text{LB}^*$.

[2]All these instances are available online at `http://mobiz.vives.be/inrc2/`

$\lvert\mathcal{N}\rvert$	$M = 4$				$M = 8$			
	instance	LB*	UB*	Gap (%)	instance	LB*	UB*	Gap (%)
30	n030w4_1_6-2-9-1	1615	1685	4.3	n030w8_1_2-7-0-9-3-6-0-6	1920	2070	7.8
	n030w4_1_6-7-5-3	1740	1840	5.7	n030w8_1_6-7-5-3-5-6-2-9	1620	1735	7.1
35	n035w4_0_1-7-1-8	1250	1415	13.2	n035w8_0_6-2-9-8-7-7-9-8	2330	2555	9.7
	n035w4_2_8-8-7-5	1045	1145	9.6	n035w8_1_0-8-1-6-1-7-2-0	2180	2305	5.7
40	n040w4_0_2-0-6-1	1335	1640	22.8	n040w8_0_0-6-8-9-2-6-6-4	2340	2620	12.0
	n040w4_2_6-1-0-6	1570	1865	18.8	n040w8_2_5-0-4-8-7-1-7-2	2205	2420	9.8
50	n050w4_0_0-4-8-7	1195	1445	20.9	n050w8_1_1-7-8-5-7-4-1-8	4625	4900	5.9
	n050w4_0_7-2-7-2	1200	1405	17.1	n050w8_1_9-7-5-3-8-8-3-1	4530	4925	8.7
60	n060w4_1_6-1-1-5	2380	2465	3.6	n060w8_0_6-2-9-9-0-8-1-3	1970	2345	19.0
	n060w4_1_9-6-3-8	2615	2730	4.4	n060w8_2_1-0-3-4-0-3-9-1	2260	2590	14.6
70	n070w4_0_3-6-5-1	2280	2430	6.6	n070w8_0_3-3-9-2-3-7-5-2	4400	4595	4.4
	n070w4_0_4-9-6-7	1990	2125	6.8	n070w8_0_9-3-0-7-2-1-1-0	4540	4760	4.8
80	n080w4_2_4-3-3-3	3140	3320	5.7	n080w8_1_4-4-9-9-3-6-0-5	3775	4180	10.7
	n080w4_2_6-0-4-8	3045	3240	6.4	n080w8_2_0-4-0-9-1-9-6-2	4125	4450	7.9
100	n100w4_0_1-1-0-8	1055	1230	16.6	n100w8_0_0-1-7-8-9-1-5-4	2005	2125	6.0
	n100w4_2_0-6-4-6	1470	1855	26.2	n100w8_1_2-4-7-9-3-9-2-8	2125	2210	4.0
110	n110w4_0_1-4-2-8	2210	2390	8.1	n110w8_0_2-1-1-7-2-6-4-7	3870	4010	3.6
	n110w4_0_1-9-3-5	2255	2525	12.0	n110w8_0_3-2-4-9-4-1-3-7	3375	3560	5.5
120	n120w4_1_4-6-2-6	1790	2165	20.9	n120w8_0_0-9-9-4-5-1-0-3	2295	2600	13.3
	n120w4_1_5-6-9-8	1820	2220	22.0	n120w8_1_7-2-6-4-5-2-0-2	2535	3095	22.1
	Average 4 weeks			12.6	Average 8 weeks			9.1

Table 8: Best results obtained on the benchmark.

6.2. *Numerical results*

The tests were all performed on a single thread of an Intel(R) Core(TM)i7-3770 CPU @ 3.40GHz processor. Our implementation calls only free third-party softwares (not only free for academics, but also for potential industrial users). We use the branch-and-price framework BCP in which the chosen linear solver is CLP[3], and the subproblems are solved with the resource constrained shortest path from the Boost library[4]. We also conducted comparative tests using other linear solvers (e.g., CPLEX or Gurobi) but none of them was significantly better, which gives another motivation for the choice of the open-source option. The source code of the software, and the parameter files corresponding to the tests described below are shared on the git repository `https://github.com/jeremyomer/StaticNurseScheduler` under an open licence.

The figures reproduced in this section represent the repartitions of integrality gaps and computational times using Tukey boxplots: the bottom and top of a box are the first and fourth quartile, the band inside a box is the median and the ends of the whiskers are the highest (lowest) values within 1.5 interquartile from the top (bottom) of the box (see [10] for a more detailed description). In Figures 6 and 7, $\textsc{Gap}^0 = (\mathrm{UB}^0 - \mathrm{LB}^*)/\mathrm{LB}^*$ designates the gap of the initial solution, where UB^0 is the best upper bound at the end of the initialization. Similarly, $\textsc{Gap} = (\mathrm{UB} - \mathrm{LB}^*)/\mathrm{LB}^*$ designates the gap of the best solution obtained after the LNS has been run. We refer as Imp to the *improvement* obtained by applying the LNS to the initial solution, i.e., $\textsc{Imp} = (\textsc{Gap}^0 - \textsc{Gap})/\textsc{Gap}^0 = (\mathrm{UB}^0 - \mathrm{UB})/(\mathrm{UB}^0 - \mathrm{LB}^*)$.

For a better evaluation of the ALNS, it is necessary to have some time left after the initialization, so we set a time limit higher than that of the INRC–II. We also use a formula that depends on the number of nurses and weeks to compute the time limit, but we settled for $M \times [60 + 6\,\lvert\mathcal{N}\rvert]$ seconds instead of the $M \times [10 + 3\,(\lvert\mathcal{N}\rvert - 20)]$ seconds suggested in the INRC–II. Finally, under label t_{init} is the *percentage* of the allowed time spent in the initialization procedure; the rest of the time is dedicated to the improvement of the initial solution by the ALNS.

Besides the numerous tests presented in the following sections, we also run the branch-and-price without any heuristic improvement (e.g., no ALNS nor rolling horizon) with a much larger time limit (24 hours). We did that to see if the branch-and-price itself was able to reach optimality with sufficient time. The best

[3]BCP and CLP are part of the COIN-OR project. They are available, respectively, at `http://www.coin-or.org/projects/Bcp.xml` and `http://www.coin-or.org/Clp/`

[4]The boost graph library is available at `http://www.boost.org/doc/libs/1_61_0/libs/graph/doc/index.html`

lower bounds reported in Table 8 (under "LB^*") were all obtained from these 24 hours executions, but it never produced the best upper bound ("UB^*"). Overall, optimality could be proved only for smaller test instances with up to 21 nurses and a four weeks planning.

6.2.1. Influence of the control period on the rolling horizon initialization

In Figure 6, we study the influence of the length of the control horizon on the performance of the algorithm. We ran the software for all possible values, i.e., one to four weeks for the 4-weeks instances, and one to eight weeks for the 8-weeks instances. At each iteration of the rolling horizon procedure, the problem is solved with branch and price until optimality is reached or until two successive executions of the primal heuristic of Section 4.3.4 provide no improvement in the upper bound. The sampling horizon is equal to one week in all our tests.

First, from Figures 6a and 6b, one sees that short control horizons do not yield good initial solutions. This can be explained as follows. When the control horizon is short, only a few variables are constrained to be integer during each solution step. Therefore, when the horizons are shifted towards the future, the variables that were relaxed and become integer may take very different values from the (fractional) ones they had in the previous step. This may induce a loss of quality that reflects a bad anticipation when the control horizon is too short. This is particularly true because of the succession constraints **H3**. From Figures 6c and 6d, one sees that the best results are obtained with rolling horizons of 3 and 8 weeks for 4-weeks and 8-weeks instances, respectively. Figures 6e and 6f show that the longer the control horizon, the more time is spent in the initialization. This meets expectations since many more variables are constrained to be integer when the length of the control period increases, thus making these problems harder to solve.

6.2.2. Performance of the initialization method

In Figure 7, we study the impact of the initialization method. We consider four methods for obtaining an initial solution. In FEASIBLE, 2-DIVES and REPEAT, the initial solution is obtained by running the branch-and-price procedure and stopping it, respectively, after the first feasible solution is obtained, after the primal heuristic of Section 4.3.4 has been run twice, and after two successive executions of the primal heuristic of Section 4.3.4 provide no improvement in the upper bound. In the ROLLING strategy, the rolling-horizon procedure is applied with the control horizon length that gave the best results in the previous section. The ALNS is then run with these initial solutions.

In Figures 7a–7d, we observe that the best method is the rolling horizon method, both in terms of quality of the initial solution (GAP^0), and of the solution obtained after the ALNS is run (GAP).

From figures 7e and 7f, one sees that much more time is spent in the initialization for the REPEAT and ROLLING strategies, particularly on the 8-weeks instances. Given the quality of the corresponding solutions, the larger time spent for a good initialization is obviously worth the loss of time spent in the ALNS.

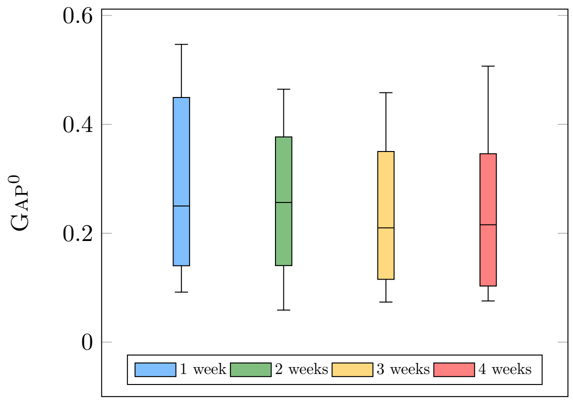
6.2.3. Influence of the destruction operator on the LNS

In this Section, we compare the destruction operators of the ALNS that are presented in Section 5.1. The results are displayed in Figures 8a–8d, where we focus on the best upper bound found by the solution algorithm.

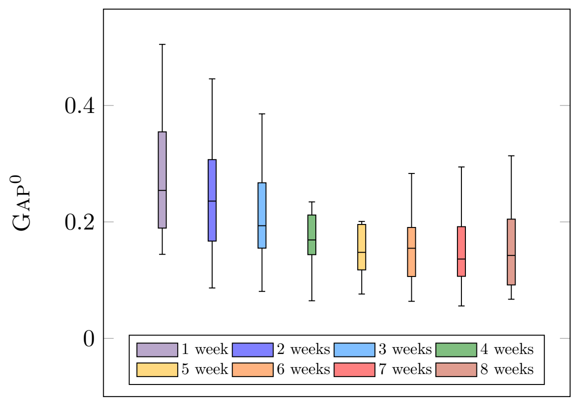
On Figures 8a–8b, we study the impact of the total number of schedule weeks destroyed ($N_N \times N_W$) at each iteration of the ALNS. For this, we compare five values evenly spread from 32 to 96 weeks. We do not observe a significant impact of the total number of weeks destroyed. In all the other tests (in this section and in the previous ones), we set $N_N \times N_W = 48$, which seems to be the best if we consider both 4-weeks and 8-weeks instances.

On Figures 8c–8d, we report the comparison of the following ALNS strategies:

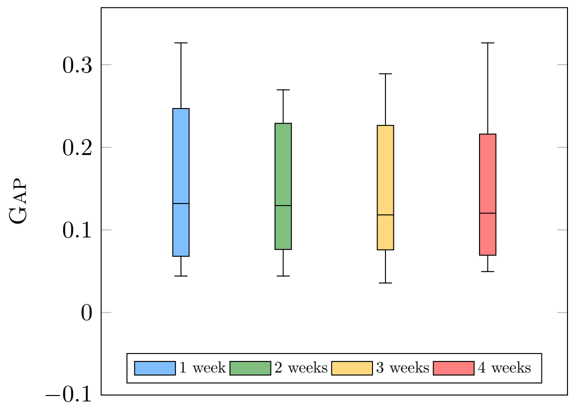
- only partial schedules are destroyed (label: "no partial"),
- only complete schedules are destroyed (label: "only partial"),
- selected nurses always have the same type (label: "`D_Type`"),



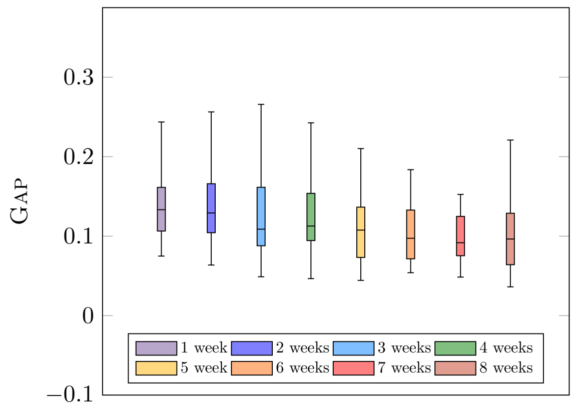
(a) 4-weeks instances.



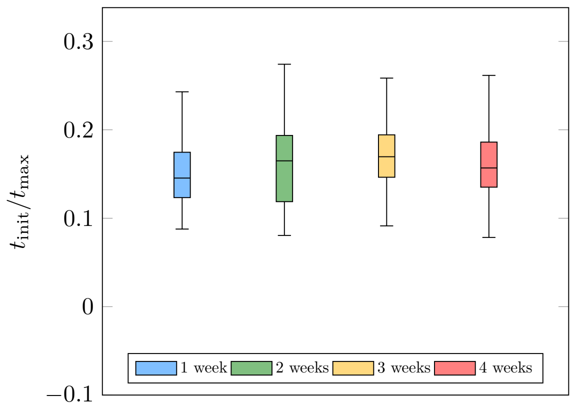
(b) 8-weeks instances.



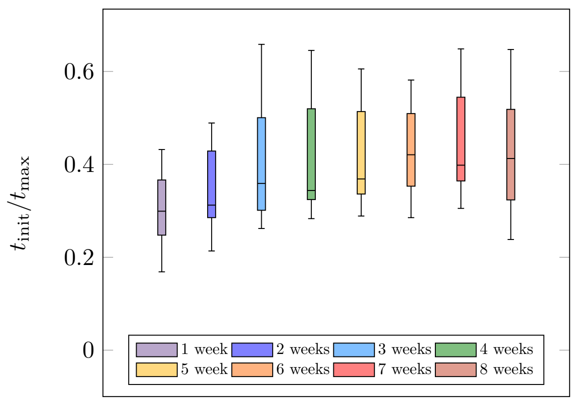
(c) 4-weeks instances.



(d) 8-weeks instances.



(e) 4-weeks instances.

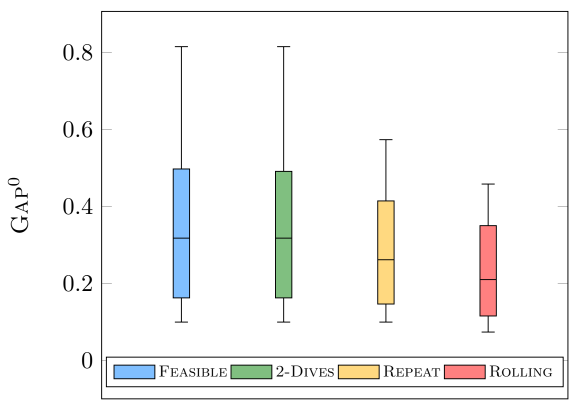


(f) 8-weeks instances.

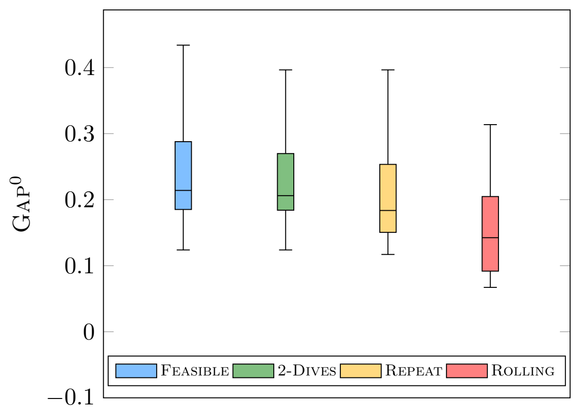
Figure 6: Impact of the size of the control horizon on the rolling-horizon initialization

- selected nurses always have the same contract (label: "`D_Contract`"),
- selected nurses are always picked randomly among all the nurses (label: "`D_Random`").

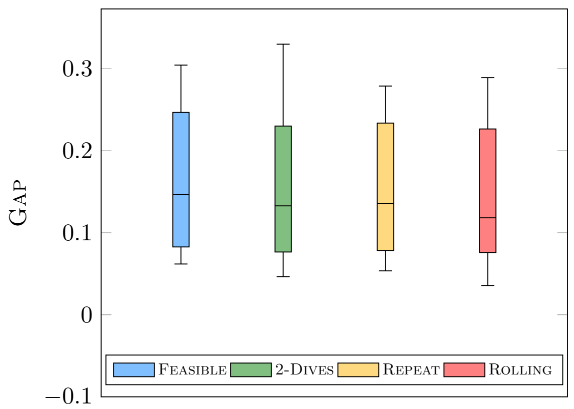
In the first two strategies, every destruction operator is allowed for the nurse selection, and in the last three strategies the schedules can either be completely or partially destroyed. As a reference, we also represent



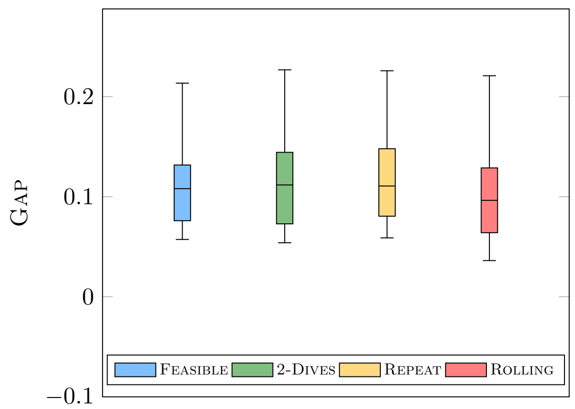
(a) 4-weeks instances.



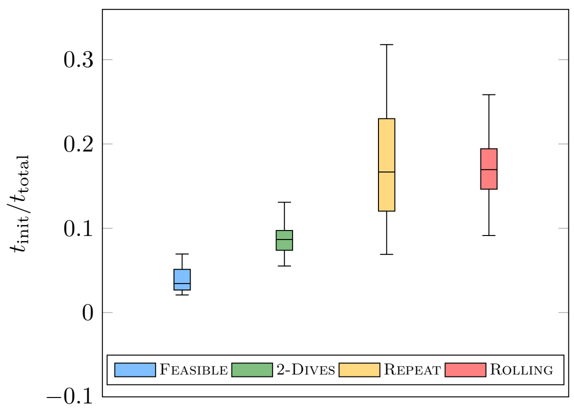
(b) 8-weeks instances.



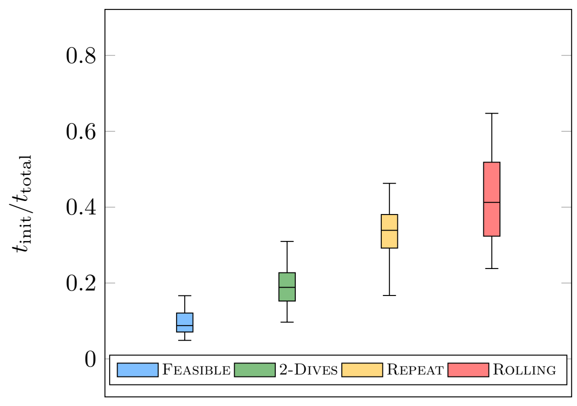
(c) 4-weeks instances.



(d) 8-weeks instances.



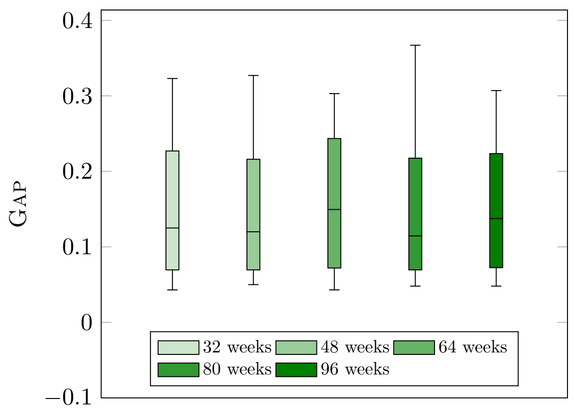
(e) 4-weeks instances.



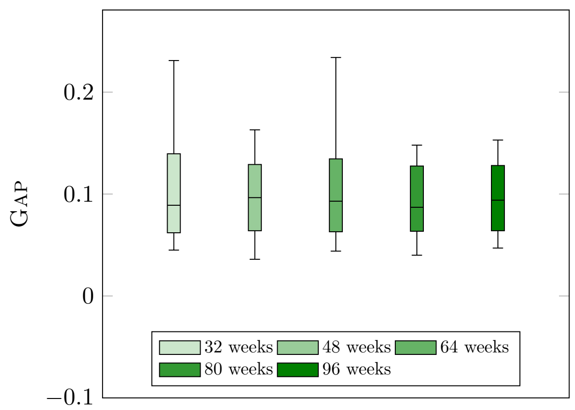
(f) 8-weeks instances.

Figure 7: Comparison of the initialization methods

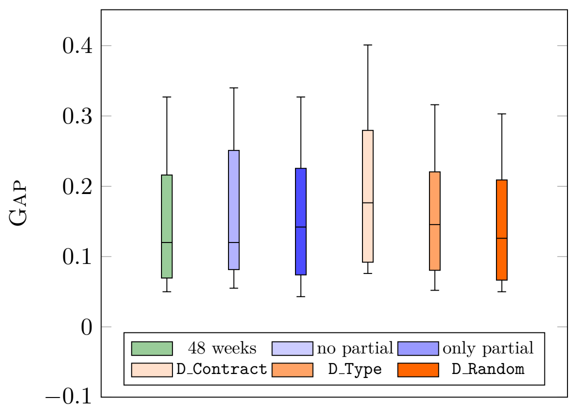
the integral gap of the ALNS where every destruction operator is used in the roulette-whell procedure(label: ”48 weeks”). The results first show that there is a significant loss in performance if the partial destruction of schedules is not allowed, whereas the opposite is not true if schedules are only partially destroyed. Second, we observe that the performance is even more sensitive to the selection strategy of the free nurses. Overall, there is a benefit in considering the three strategies in the ALNS, but the random choice over all the nurses is significantly better than the other two “smarter” strategies. This is not an original observation in the field of metaheuristics, where randomness is sometimes the best tool towards unexpected improvements.



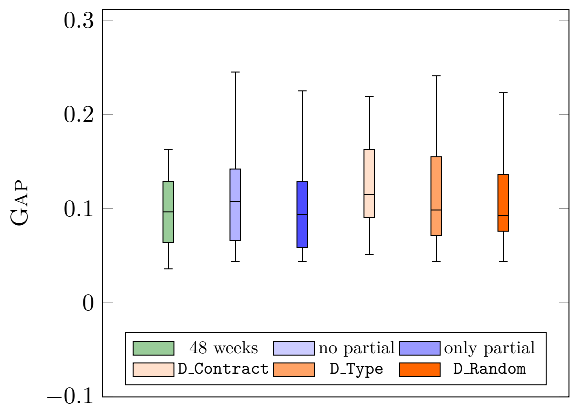
(a) 4-weeks instances.



(b) 8-weeks instances.



(c) 4-weeks instances.



(d) 8-weeks instances.

Figure 8: Performance of the ALNS depending on the destruction operator (diagrams for instances of 4 and 8 weeks).

6.2.4. Comparison with the results of the dynamic version of the problem

To conclude the tests, we compare the results described above with the results published at the end of the INRC–II, which are the only other published results for these instances. These results are for the dynamic version of the problem, where the weeks are scheduled sequentially without any information on the demand of future weeks. This comparison is summarized in Table 9. The values indicated for the dynamic problem are the best results over all the teams and all the random seeds input by the organizers. For the static version, we provide the best integer solution we found in all our tests and the solutions found by the ALNS initialized with the rolling horizon and $N_N \times N_W = 48$ weeks destroyed at each iteration. In every case, we provide the value of the best integer solution (UB^*), and for each static result, we give the relative improvement of the solution value when considering the static instead of the stochastic version of the problem (IMP). The best relative improvement can be seen as an estimation of the relative value of perfect information for these instances.

It is comforting to observe that our best solution of the static version of the problem achieve an overall 15.2% relative improvement with respect to the best solution found in the dynamic version of the problem.

The results show that the largest improvements are obtained for the 8 weeks instances, which was expected. In the dynamic version, the schedule of the first week is planned without information about future demand. It is thus logical that the largest errors due to uncertainty are made for the largest planning horizon. Finally, we still observe that for three small instances (n040w4_2_6-1-0-6, n050w4_0_0-4-8-7 and n050w4_0_7-2-7-2), the ALNS does not achieve to improve the best dynamic solution. The reason might be that these instances were part of the first phase of the competition where the participants could choose the random seeds that provided the best results among all their tests. As a consequence, these results are the best over thousands of runs of stochastic algorithms. In contrast, the results of the ALNS reflect only one specific execution of the algorithm.

	dynamic	static: best		static: ALNS	
instance	UB*	UB*	Imp(%)	UB*	Imp(%)
n030w4_1_6-2-9-1	1755	1685	4.2%	1695	3.5%
n030w4_1_6-7-5-3	1935	1840	5.2%	1890	2.4%
n035w4_0_1-7-1-8	1630	1415	15.2%	1425	14.4%
n035w4_2_8-8-7-5	1255	1145	9.6%	1155	8.7%
n040w4_0_2-0-6-1	1730	1640	5.5%	1685	2.7%
n040w4_2_6-1-0-6	1880	1865	0.8%	1890	-0.5%
n050w4_0_0-4-8-7	1490	1445	3.1%	1505	-1.0%
n050w4_0_7-2-7-2	1480	1405	5.3%	1500	-1.3%
n060w4_1_6-1-1-5	2815	2465	14.2%	2505	12.4%
n060w4_1_9-6-3-8	2950	2730	8.1%	2750	7.3%
n070w4_0_3-6-5-1	2700	2430	11.1%	2435	10.9%
n070w4_0_4-9-6-7	2430	2125	14.4%	2175	11.7%
n080w4_2_4-3-3-3	3535	3320	6.5%	3340	5.8%
n080w4_2_6-0-4-8	3570	3240	10.2%	3260	9.5%
n100w4_0_1-1-0-8	1445	1230	17.5%	1245	16.1%
n100w4_2_0-6-4-6	2100	1855	13.2%	1950	7.7%
n110w4_0_1-4-2-8	2710	2390	13.4%	2440	11.1%
n110w4_0_1-9-3-5	2920	2525	15.6%	2560	14.1%
n120w4_1_4-6-2-6	2435	2165	12.5%	2170	12.2%
n120w4_1_5-6-9-8	2485	2220	11.9%	2220	11.9%
Average 4 weeks			9.9%		8.0%
n030w8_1_2-7-0-9-3-6-0-6	2340	2070	13.0%	2070	13.0%
n030w8_1_6-7-5-3-5-6-2-9	1900	1735	9.5%	1735	9.5%
n035w8_0_6-2-9-8-7-7-9-8	3020	2555	18.2%	2555	18.2%
n035w8_1_0-8-1-6-1-7-2-0	2770	2305	20.2%	2305	20.2%
n040w8_0_0-6-8-9-2-6-6-4	3310	2620	26.3%	2620	26.3%
n040w8_2_5-0-4-8-7-1-7-2	2700	2420	11.6%	2420	11.6%
n050w8_1_1-7-8-5-7-4-1-8	5410	4900	10.4%	4900	10.4%
n050w8_1_9-7-5-3-8-8-3-1	5435	4925	10.4%	4925	10.4%
n060w8_0_6-2-9-9-0-8-1-3	2765	2345	17.9%	2345	17.9%
n060w8_2_1-0-3-4-0-3-9-1	3065	2590	18.3%	2590	18.3%
n070w8_0_3-3-9-2-3-7-5-2	5115	4595	11.3%	4595	11.3%
n070w8_0_9-3-0-7-2-1-1-0	5390	4760	13.2%	4760	13.2%
n080w8_1_4-4-9-9-3-6-0-5	4995	4180	19.5%	4180	19.5%
n080w8_2_0-4-0-9-1-9-6-2	5030	4450	13.0%	4450	13.0%
n100w8_0_0-1-7-8-9-1-5-4	3080	2125	44.9%	2125	44.9%
n100w8_1_2-4-7-9-3-9-2-8	3055	2210	38.2%	2210	38.2%
n110w8_0_2-1-1-7-2-6-4-7	5155	4010	28.6%	4010	28.6%
n110w8_0_3-2-4-9-4-1-3-7	4805	3560	35.0%	3560	35.0%
n120w8_0_0-9-9-4-5-1-0-3	3615	2600	39.0%	2600	39.0%
n120w8_1_7-2-6-4-5-2-0-2	3510	3095	13.4%	3095	13.4%
Average 8 weeks			19.3%		20.6%
Average overall			15.2%		13.6%

Table 9: Best results obtained on the benchmark

7. Conclusions

This article deals with the nurse scheduling problem as described in the context of the international competition INRC–II. Our first contribution is the description of a branch-and-price algorithm procedure based on rotations, i.e., a sequence of working days preceded and followed by at least one day off. This decomposition, adapted from aircrew planning, allows to significantly reduce the complexity of the subproblems and the memory space occupied by the generated columns. We modeled the assignments of days-off and skills as flow problems which are handled in the master problem. This allows to avoid the individual assignment of skills and relax the integrality of the skill assignment variables. What is more, a strong effort has been done to accelerate the solution of the subproblems where negative reduced cost rotations are generated. The subproblems are finally modeled as shortest path problems with one resource constraint corresponding to the total number of worked days in the rotation. Based on a theoretical study of the size of the graph, we could thus infer that in practice, each subproblem (one per nurse) should be executed in linear time with respect to the length of the planning horizon if a classical labeling algorithm is used.

To achieve good results on large instances, we then described how an ALNS search could be implemented as primal heuristic in interaction with the branch-and-price. The ALNS uses several destruction operators that consider different strategies for the choice of the nurses whose schedules are destroyed and for the number of schedule weeks destroyed. We then describe several primal heuristics to initialize the ALNS, including a rolling-horizon procedure, where the weekly schedules are computed sequentially in chronological order.

Finally, the experimental tests focus on a benchmark of forty instances published in the INRC–II. The instances describe the constraints for the schedule of 30 to 120 nurses over 4 and 8-weeks horizons. The results highlight that the best initialization method is the rolling horizon procedure, even though it takes a greater fraction of the total computational time. We also carried out a sensitivity analysis of the ALNS to the choice of the destruction operators. The main conclusion being that there is indeed a benefit in using an adaptive strategy, even though a random selection of the nurses whose schedules are destroyed also achieves good results. Finally, we showed that algorithm achieves an average 15.2% improvement with respect to the best results reported during the INRC–II for the dynamic version of the problem.

Future research should aim at finding optimal solutions of instances with 30 and more nurses. For this, we think that other decompositions could be considered in the branch-and-price procedure to lower the integrality gap. One option is the classical decomposition where complete individual schedules are generated, but other more refined rotation-based decompositions could also be developed. For instance, additional layers could be added in the master problem flow network to reduce the gap due to the overwork on week-ends. Another direction of research is the adaptation of the rotation model to other constraints described in the literature.

[1] Bard, J. F., and Purnomo, H. W. Preference scheduling for nurses using column generation. *European Journal of Operational Research 164*, 2 (2005), 510 – 534.

[2] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. Branch-and-price: Column generation for solving huge integer programs. *Operations Research 46*, 3 (1998), pp. 316–329.

[3] Beliën, J., and Demeulemeester, E. A branch-and-price approach for integrating nurse and surgery scheduling. *European Journal of Operational Research 189*, 3 (2008), 652–668.

[4] Braekers, K., and Janssens, G. K. Shortest route problem with soft time windows. In *The Europeand Simulation and Modelling Conference* (2013), S. Onggo and A. Kavicka, Eds., pp. 279–283.

[5] Burke, E. K., and Curtois, T. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research 237*, 1 (2014), 71–81.

[6] Burke, E. K., De Causmaecker, P., Berghe, G. V., and Van Landeghem, H. The state of the art of nurse rostering. *Journal of Scheduling 7*, 6 (2004), 441–499.

[7] Ceschia, S., Dang, N. T. T., De Causmaecker, P., Haspeslagh, S., and Schaerf, A. The second international nurse rostering competition. In *PATAT 2014: Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling* (2014), pp. 554–556.

[8] Desaulniers, G., Desrosiers, J., and Solomon, M. M. *Column generation*, vol. 5. Springer Science & Business Media, 2006.

[9] Dumas, Y., Soumis, F., and Desrosiers, J. Optimizing the schedule for a fixed vehicle path with a convex inconvenience costs. *Transportation Science 24*, 2 (1990), 145–152.

[10] Frigge, M., Hoaglin, D. C., and Iglewicz, B. Some implementations of the boxplot. *The American Statistician 43*, 1 (1989), 50–54.

[11] Gamache, M., and Soumis, F. A method for optimally solving the rostering problem. In *Operations Research in the Airline Industry*, G. Yu, Ed., vol. 9 of *International Series in Operations Research and Management Science.* Springer US, New York, 1998, ch. 5, pp. 124–157.

[12] Garcia, R. *Resource constrained shortest paths and extensions.* ProQuest, 2009.

[13] He, F., and Qu, R. A constraint programming based column generation approach to nurse rostering problems. *Computers & Operations Research 39*, 12 (2012), 3331 – 3343.

[14] Irnich, S., Desaulniers, G., et al. Shortest path problems with resource constraints. *Column generation 6730*, 33–65.

[15] Jaumard, B., Semet, F., and Vovor, T. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research 107*, 1 (1998), 1–18.

[16] Maenhout, B., and Vanhoucke, M. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling 13*, 1 (2010), 77–93.

[17] Pisinger, D., and Ropke, S. *Handbook of Metaheuristics.* Springer US, Boston, MA, 2010, ch. Large Neighborhood Search, pp. 399–419.

[18] Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L.-M. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks 54*, 4 (2009), 190–204.

[19] Qurashi, A. G., Taniguchi, E., and Yamada, T. Column generation-based heuristics for vehicle routing problem with soft time windows. *Journal of the Eastern Asia Society for Transportation Studies 8* (2010).

[20] Qurashi, A. G., Taniguchi, E., and Yamada, T. Exact solution for vehicle routing problem with soft time windows and dynamic travel time. *Asian Transport Studies 2*, 1 (2012), 48–63.

[21] Richalet, J., Rault, A., Testud, J., and Papon, J. Model predictive heuristic control: applications to industrial processes. *Automatica 14*, 5 (1978), 413 – 428.

[22] Saddoune, M., Desaulniers, G., and Soumis, F. Aircrew pairings with possible repetitions of the same flight number. *Computers & Operations Research 40*, 3 (2013), 805 – 814. Transport Scheduling.