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], Samuel Rosat[a,d]

[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] *Univ Rennes, INSA Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France*
[d] *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 is that most constraints are soft, meaning that they can be violated at the price of a penalty. We model the problem as 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. In contrast, classical models involve individual nurse schedules over the complete horizon. 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 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. A day is divided into several time frames called *shifts*, and each nurse has a set of *skills* corresponding to tasks he/she can perform. When a nurse works on a given day, he/she is assigned to one shift with a skill he/she possesses. A day with no assignment is a *day off*, and a sequence of days off preceded and followed by assignments is a *rest period*. The NSP then consists in building a *roster* for each nurse, i.e., a sequence of assignments and rest periods covering the planning horizon. Most rostering problems aim at balancing the work among employees (i.e., each employee should receive a fair amount of worked days). As a consequence the cost

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

of the complete schedule usually aggregates an evaluation of the global quality of service it provides with penalties associated to unbalanced personalized rosters. A fair deal of the complexity of the NSP can be attributed to the personalized aspect of the schedules: the same schedule may indeed have very different costs if given to one nurse or another, because they have different contracts or preferences on days off. In contrast, workforce scheduling problems usually build anonymous rosters thus making the problem easier. The NSP has been widely studied for more than two decades. The reader is referred to [11] or [7] 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 rosters. Most constraints are 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-cut. 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 non-restricted version. The reader can find an overview of column-generation algorithms in [12] and a comprehensive review on SPPRC in [22].

To the best of our knowledge, the first published attempt to tackle the NSP by means of branch-and-price dates back from 1998 [23]. The authors develop a generic framework that handles a large variety of 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 [26] 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. In [6], 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 pricing problem. They report excellent results on the benchmark of the first international nurse rostering competition (INRC) [20]. The pricing problem can also be solved with constraint programming, as in [21]. Gomes *et al.* [19] enrich their branch-and-price scheme with a variable neighborhood search to tackle instances from the second international nurse rostering competition (INRC-II) [8]. They use a classical decomposition by roster for each nurse, but they report results only for the 4-weeks instances. Another related study is reported in [3], where the authors treat nurse and surgery scheduling by branch-and-price in an integrated approach without considering the personalized aspect of the problem.

Several authors used *branch-and-price* with SPPRC to solve other rostering problems than NSPs. Personalized rosters are also encountered in airline crew scheduling problems. However, the rotations (i.e., a sequence of flights that starts and ends at the same base) are usually pre-computed, and the rostering problem consists in choosing a sequence of rotations to build a roster for each crew member (see [24] for more details). In [16], the authors solve large scale instances from Air France. In [18], the authors propose four different algorithms to solve scheduling problems with multi-skills and heterogeneous workforce. In contrast, Boyer *et al.* [4] model the pricing problem with a formal grammar that allows the representation of richer constraints, and solve them with dynamic programming. This allows the authors to tackle a general personalized workforce scheduling problem that is more complex than the considered NSP. In particular, they build the detailed sequence of tasks each employee performs on one day. However, the computational price is such that they must restrict to one-day schedules for up to 50 employees.

Finally, among exact approaches that are not based on brand-and-price, Santos *et al.* [34] present a MIP for the instances of the first INRC and several ways to improve the bounds (especially, the dual ones). They

prove optimality for several instances.

1.2. Main contributions

In this article we deal with a static version of the NSP described in the INRC–II [8]. Our method is 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 proportion of hard constraints. Other constraints can be violated at the price of a penalty. As a consequence, in a column-generation context, the number of generated columns increases, and fewer labels can be removed in the sub-problems. Our solution approach 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 problem generates complete rosters and decision variables indicate whether these complete rosters are to be used in the solution. In our work, we do not generate complete but partial rosters, that are called *rotations*[1]. A rotation is a sequence of shift assignments on consecutive days that are preceded and followed by at least one day off; it does not specify the skill used by the nurse on each assignment. The IP then builds complete schedules for the nurses from the existing rotations, and allocates the skills that they must perform. Personalized rotations are 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. We investigate this approach, because it reduces the number of feasible columns that can be generated and it decreases the complexity of the pricing problem. This change of paradigm also leads to the development of specific branching rules (Section 4.3) and to structural modifications in the pricing network (Section 5.3). The separation of the allocations of shifts and skills is independent from the partial-roster approach. This decomposition, as well as the aggregation of nurses in skill assignment (Section 5.2) 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). The ALNS is a local search where the iterates are obtained by sequentially re-optimizing the problem over a subset of variables while keeping the others unchanged. In our implementation, we either re-optimize complete schedules of a limited number of nurses or partial schedules of a larger number of nurses.

We also develop several primal heuristics based on our branch-and-price procedure to find the initial solution of the ALNS. One of them is a rolling-horizon method which sequentially computes weekly schedules while taking into account an estimation of the impact of current decisions on the future.

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[2] for reproduction of the results, future comparisons, improvements and extensions.

Our tests are based on the instances of the INRC–II. They involve schedules of 30 to 120 nurses over a planning horizon of four to eight weeks. The nurses can have four different skills and each day is divided into four shifts. We conduct an experimental comparison of several initialization methods for the ALNS and study the sensitivity to the choice of the 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%; the ALNS lowers this value to an average 10.9% gap. Finally, we compare our results to the best solutions found in the INRC–II, and to three other published studies. The comparison confirms that our method is able to produce good solutions in a reasonable computational time for all the instances.

[1] This denomination comes from the airline industry where similar modelling is commonly used (see [15]).
[2] The code implementing the methods described in this article is publicly shared on the Git repository [25]

1.3. Organization of the paper

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 describe the branch-and-price algorithm and the pricing problem. We then specify the corresponding implementation choices in Section 5. The ALNS procedure and the rolling-horizon method that finds the initial solution are expounded in Section 6. In Section 7, we report numerical results that demonstrate the relevance of our approach. Section 8 provides concluding remarks.

2. Description of the problem

We consider the NSP proposed by Ceschia *et al.* [10]. While the problem is stated as a dynamic one in [10], where only a part of the information is given at each stage of the solution process, here we focus on its static version where complete 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 shifts are respectively denoted Σ and $\mathcal{S}$. For the sake of readability, indices are standardized in the following way: nurses are denoted as $i \in \mathcal{N}$, days as $k \in \{1, \ldots, K\}$, shifts as $s \in \mathcal{S}$ and skills as $\sigma \in \Sigma$. Finally, the pair $p = (k, s)$ denotes the worked shift s of day k, and is called *assignment* p. We summarize all other data 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 days off for nurse i
WE_i^+	max. number of worked week-ends over the planning horizon for nurse i
Π_i	set of assignments p 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	
$D_{p\sigma}$	min. demand in nurses performing skill σ on assignment p
$O_{p\sigma}$	optimal demand in nurses performing skill σ on assignment p

Table 1: Summary of the input data.

Remark (Initial state)**.** *For practical reasons, it is necessary that the model 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 are straightforward and of no particular interest for the reader.*

We do not intend to handle every constraint that can be found in the literature on NSPs; we choose the set of constraints proposed by the organizers of the INRC–II in [10] 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 cannot 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 main feature of this benchmark is that most constraints are soft. All 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_{SX}. For constraint **S2**, the unit weights for consecutive working days and consecutive shift are respectively denoted c_{S2a} and c_{S2b}.

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 assignment p must be at least equal to the minimum demand $D_{p\sigma}$.
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 assignment p must be at least equal to the optimal demand $O_{p\sigma}$. Each missing nurse is penalized according to the unit weight.
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 undesired assignment $p \in \Pi_i$ of a nurse i is penalized by the unit weight.
S5	**Complete week-end**: Some nurses 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 penalized by the unit weight.
S6	**Total assignments**: For each nurse i, the total number of assignments must be within $[L_i^-, L_i^+]$. The difference (in either direction) is penalized by the unit weight.
S7	**Total working week-ends**: For each nurse i, the number of week-ends with at least one assignment must be less than or equal to WE_i^+. The number of worked week-ends over that limit is penalized by the unit weight.

Table 2: Constraints handled by the software.

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: this sub-problem is formulated as a flow model in Section 3.2. The complete model described in Section 3.3 includes these flow constraints, the skills allocation and the constraints relative to the complete planning horizon (**H2**, **S1**, **S6** and **S7**).

3.1. Rotations: definitions and notations

To formulate our mathematical model for the NSP, we first specify some vocabulary in our context and introduce notation. A *rotation* is a list of assignments on consecutive days, preceded and followed by at least one day off. It is important to note that a rotation does *not* contain any information about the skills performed on these assignments. 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 corresponds only to the penalties associated with the soft constraints **S2**, **S4** and **S5**. The penalties associated with the other soft constraints can only be computed for complete rosters.

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

Day	*0*	*1*	*2*	*3*	*4*	*5*	*6*
Shift	*Early*	*Day*	*Off*	*Off*	*Night*	*Night*	*Off*
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.

3.2. A flow model for the creation of individual rosters

In this section, we describe how we build a *skill-less* roster over the complete planning horizon from the set Ω_i of feasible rotations for nurse $i \in \mathcal{N}$. By skill-less roster, we mean that it does not specify which skill the nurse performs on the days he/she works. The problem of building the skill-less roster of a given nurse i can be formulated as a shortest path problem in a weighted and directed graph called *rostering graph* of nurse i.

In essence, a vertex of the rostering graph corresponds to a day and an arc corresponds either to a feasible rotation or to a rest period. Since the cost of a rest period depends only on its length, there is at most one rest arc between each pair of vertices. In contrast, there can be a large number of parallel rotation arcs that will correspond to distinct sequences of assignments. A path from the source to the sink 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 in Table 3.

$\mathcal{V}_i$	*source, sink*	One source node o_i, one sink node d_i.
	rest nodes	For each day k, one rest node $R_{i,k}$.
	work nodes	For each day k, one work node $W_{i,k}$.
$\mathcal{A}_i$	*rotation arcs*	For each rotation $j \in \Omega_i$ beginning on day b and ending on day e, one arc $(W_{i,b}, R_{i,e})$ with cost c_j (**S2** + **S4** + **S5**) is added. When several rotations share the same starting and ending days, parallel arcs are added.
	min. rest arcs	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 $(R_{i,k}, W_{i,l})$ is added with cost $\max\{0, c_{\mathsf{S3}}\left(\mathrm{CR}_i^- - (l-k)\right)\}$ (min. consecutive days off penalty associated with **S3**).
	max. rest arcs	For each day $k \in \{\mathrm{CR}_i^-, \dots, (K-1)\}$, one arc $(W_{i,k}, W_{i,(k+1)})$ with cost c_{S3} is added. These arcs are only used when the maximum consecutive number of days off CR_i^+ is exceeded.
	artificial arcs	Arcs from the source to $R_{i,1}$ and $W_{i,1}$, and arcs from $R_{i,K}$ and $W_{i,K}$ 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 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, one graph is built for each nurse, so they do not include the linking staffing constraints (**H2**, **S1**).

These two remaining groups must be added to the model on top of the flow constraints (see Section 3.3). An example of rostering graph is given in Figure 1 for a 7-days planning horizon and $\mathrm{CR}_i^+ = 4$.

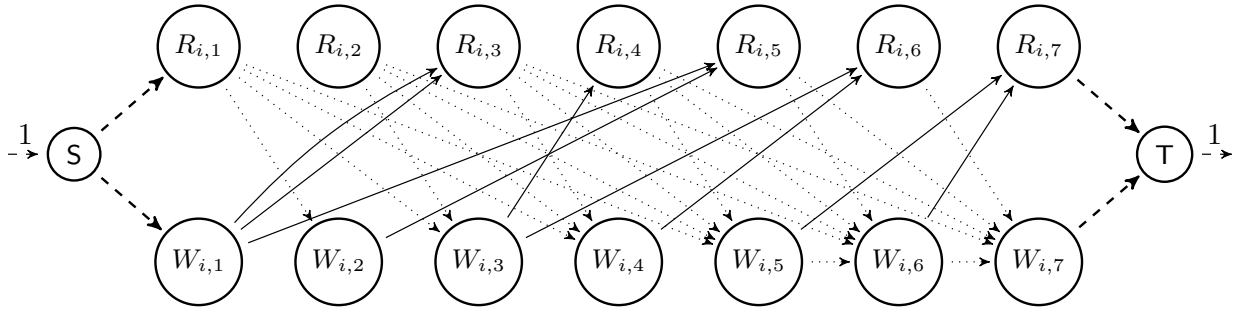


Figure 1: Example of a rostering graph for nurse $i \in \mathcal{N}$ over an horizon of $K = 7$ days, where the maximum number of consecutive rest days is $\mathrm{CR}^+ = 4$. The rotation arcs are the plain arrows, the rest arcs are the dotted arcs, and the artificial arcs are the dashed arrows.

3.3. Integer Programming formulation

In this section, we summarize our IP for the NSP, based on the rotations previously defined. The IP is the constrained flow model summarized in Formulation (1). Unless stated otherwise, the indices belong to the following sets: $i \in \mathcal{N}$, $p \in \{1, \ldots, K\} \times \mathcal{S}$ and $\sigma \in \Sigma$. In the rostering graph of nurse i, $\mathcal{A}_{iv}^-$ and $\mathcal{A}_{iv}^+$ include all the arcs respectively entering and leaving vertex v. The parameters and variables used in the IP are described in Table 4, where we use a bold font to denote vectors. The dual variables associated with the constraints where $\mathbf{x}$ appears are needed in the rotation-generation procedure, so we indicate them with Greek letters between brackets (α, β and δ).

Parameters		
$\mathbf{c}$		vector containing the cost of each arc in the rotation graph of each nurse
$\mathbf{c}_w$		vector of unit costs for violations of **S6**–**S7**, namely $\mathbf{c}_w = (c_{\mathsf{S6}}, c_{\mathsf{S6}}, c_{\mathsf{S7}})$
$\mathbf{a}_j$		Boolean vector indicating which assignment is in rotation j of nurse i
$\mathbf{b}_j$		vector containing the number of assignments and worked weekends in rotation j of nurse i
$\mathbf{B}_i$		vector containing the bounds associated with **S6**–**S7**, namely $\mathbf{B}_i = (L_i^-, L_i^+, \mathrm{WE}_i^+)$
Variables		
x_{ij}	$\in \{0,1\}$	$= 1$ if and only if the rotation or resting period associated with arc j is part of the roster of nurse i
$\mathbf{w}_i$	$\in \mathbb{N}^3$	vector containing the violations of the soft constraints **S6**-**S7** in the roster of nurse i
$y_{ip\sigma}$	$\in \{0,1\}$	$= 1$ if and only if nurse i performs skill σ on assignment p
$z_{p\sigma}$	$\in \mathbb{N}$	violation of the optimal demand constraint for assignment p and skill σ

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

$$\min \quad \underbrace{\mathbf{c}^T\mathbf{x}}_{\mathbf{S2},\mathbf{S3},\mathbf{S4},\mathbf{S5}} + \underbrace{\sum_i \mathbf{c_w}^T\mathbf{w_i}}_{\mathbf{S6},\mathbf{S7}} + \underbrace{c_{\mathsf{S1}} \sum_{l,\sigma} z_{l\sigma}}_{\mathbf{S1}} \tag{1a}$$

$$\text{s.t.:} \quad [\mathbf{H1}] \quad \sum_{j \in \mathcal{A}^+_{io_i}} x_{ij} = 1, \qquad \forall i \quad [\alpha_{io_i}] \tag{1b}$$

$$[\mathbf{H1}, \mathbf{H3}] \quad \sum_{j \in \mathcal{A}^+_{iv}} x_{ij} - \sum_{j \in \mathcal{A}^-_{iv}} x_{ij} = 0, \qquad \forall i, \forall v \in \mathcal{V}_i \setminus \{o_i, d_i\} \quad [\alpha_{iv}] \tag{1c}$$

$$[\mathbf{S6}, \mathbf{S7}] \quad \sum_{j \in \Omega_i} \mathbf{b_j} x_{ij} + \mathbf{w_i} \leq \mathbf{B_i}, \qquad \forall i \quad [\boldsymbol{\beta}_i] \tag{1d}$$

$$[\mathbf{H4}] \quad \sum_{j \in \Omega_i} \sum_p a_{jp} x_{ij} - \sum_{\sigma \in \Sigma i} y_{ip\sigma} = 0, \qquad \forall i, p \quad [\delta_{ip}] \tag{1e}$$

$$[\mathbf{H2}] \quad \sum_{i \in \mathcal{N}_\sigma} y_{ip\sigma} \geq D_{p\sigma}, \qquad \forall p, \sigma \tag{1f}$$

$$[\mathbf{S1}] \quad \sum_{i \in \mathcal{N}_\sigma} y_{ip\sigma} + z_{p\sigma} \geq O_{p\sigma}, \qquad \forall p, \sigma \tag{1g}$$

$$x_{ij}, y_{ip\sigma} \in \{0, 1\}, \qquad \forall i, j, p, \sigma \tag{1h}$$

$$\mathbf{w}, \mathbf{z} \geq 0 \tag{1i}$$

The objective function (1a) reflects the penalties incurred for violations of the soft constraints. It is the sum of three terms: the cost $\mathbf{c}^T\mathbf{x}$ of the chosen arcs (either rest or work arcs), the penalties $\sum_i \mathbf{c}_w^T\mathbf{w}$ for violating the constraints on the total numbers of assignments and worked week-ends (**S6**, **S7**), and the penalties $c_{\mathsf{S1}} \sum_{l,\sigma} z_{l\sigma}$ for insufficient staffing (**S1**). Constraints (1b)–(1c) model the flow conservation in the rostering graph of each nurse i. Constraints (1d) stand for the requirements involving several different rotations of each nurse i. Constraints (1e) ensure that a skill σ can be used by a nurse i only if this nurse has this skill and is working on assignment p. Constraints (1f) ensure that there are enough nurses (i.e., the minimum demand) working with skill σ on assignment p, whereas Constraints (1g) check how many nurses are missing to meet optimal demand.

It can be noted that constraints (1e)-(1g) could be represented with flow model where $\mathbf{y}$ would be the arc flows and $\mathbf{x}$ an input. As a consequence, the integrality of $\mathbf{y}$ can be deduced from that of $\mathbf{x}$. What is more, the integrality of $\mathbf{w}$ and $\mathbf{z}$ is a direct consequence of that of $\mathbf{x}$, $\mathbf{y}$ and of the input data. This justifies that branching rules can focus on $\mathbf{x}$ (see Section 4.3).

Remark. *Constraints* (1d) *are formulated in an abstract way to allow for generalization to different constraints than those considered in INRC–II. To preserve the branch-and-price approach where the pricing problem is an SPPRC, it is necessary that all the components of vectors* $\mathbf{b}_j$ *can be counted in the SPPRC by some arcs or by a resource.*

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.* [12]. In Section 4.1, we describe the overall column generation scheme that we implement to solve the linear relaxation of Formulation (1). In Section 4.2, we describe our model for the pricing problem. 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 (1) 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 pricing problem 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 (1).

The resulting decomposition of the constraints corresponding to our IP is summarized in Table 5. The pricing generate new rotations, while the master problem implements Formulation (1). 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 pricing, no rotation with two assignments on the same day should be constructed.

Master problem	Pricing problem
H1 Single assignment per day	**H1** Single assignment 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 problem does not include any constraint linking the nurses with one another, so it can be decomposed into as many independent subproblems as the number of nurses. In this section, we describe how the pricing associated to a given nurse i can be modeled as an SPPRC in a directed network.

The hard constraints of the pricing problem are enforced by the structure of the network, which is similar to others met in scheduling applications. For each possible assignment (k, s), one node $A_{k,s}$ is created and one arcs links two assignments on consecutive days A_{k,s_1} and A_{k+1,s_2} whenever (s_1, s_2) is not a forbidden shift succession (i.e. $(s_1, s_2) \notin \bar{\mathcal{F}}$). Two artificial source and sink nodes are then added, together with one arc from the source to each assignment node, and one arc from each assignment node to the sink. A small example is given in Figure 2.

By construction of the network, any path from the source to the sink corresponds to a feasible rotation. Provided that the arc costs can be such that the cost of a path is the reduced cost of the associated rotation, a feasible rotation of lowest reduced cost is then a shortest path from the sink to the source. For this, first consider a rotation j starting on day b and ending on day e. The rotation is composed of a sequence $(p_b, \ldots, p_e)$ of assignments including (at least partially) n_j weekends. Referring to Formulation 1, we compute the reduced cost of j as

$$\bar{c}_j = \underbrace{c_j}_{\mathbf{S2},\mathbf{S4},\mathbf{S5}} - \underbrace{\left(\alpha_{i,W_{i,b}} - \alpha_{i,R_{i,e}}\right)}_{\text{(1c)}} - \underbrace{\left(\left(\beta_{i,1} + \beta_{i,2}\right)(e - b + 1) + \beta_{i,3} n_j\right)}_{\text{(1d)}} - \underbrace{\sum_{l=b}^{e} \delta_{i,p_l}}_{\text{(1e)}}, \tag{2}$$

where $\beta_{i,1}$ and $\beta_{i,2}$ are the dual variables of the constraints on the total number of assignments and $\beta_{i,3}$ is that of the constraint on the total number of worked weekends.

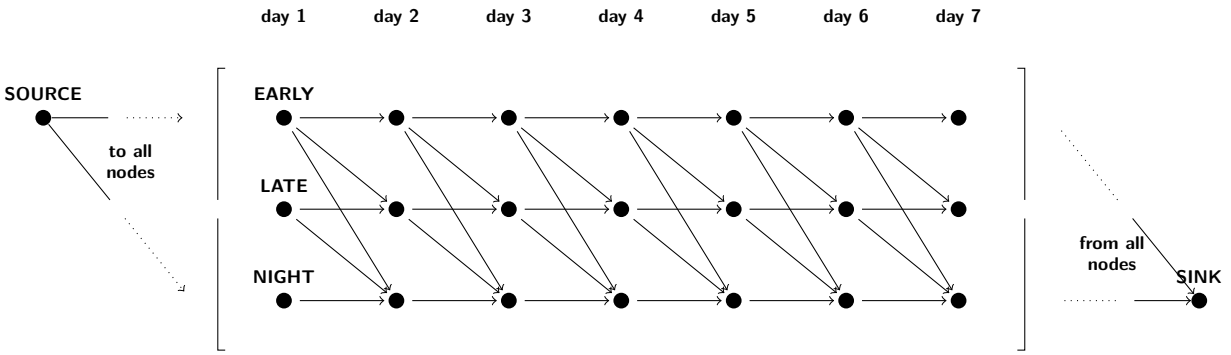


Figure 2: Example of a pricing network. Here, $K = 7$, $\mathcal{S} = \{\text{EARLY}, \text{LATE}, \text{NIGHT}\}$ and the forbidden successions are $\bar{\mathcal{F}} = \{(\text{LATE}, \text{EARLY}), (\text{NIGHT}, \text{EARLY}), (\text{NIGHT}, \text{LATE})\}$.

Among the terms that define $\bar{c}_j$ in (2), the soft penalties **S4**, **S5** and all dual costs can be added directly as arc costs to the pricing network. Indeed, these costs can either be associated with one specific assignment or they impact every assignment equally. For instance, the costs associated with the non-respect of the nurse's preferences (**S4**) result in the addition of a cost c_{S4} on all arcs whose endpoint is an undesired assignment. Similarly, the dual costs $(\beta_{i,1} + \beta_{i,2})$ and $\delta_{i,p}$ must be paid for every assignment p in the rotation, so they are simply added to the costs of the arcs with this assignment as endpoint.

In contrast, the penalties associated with the constraints on consecutive assignments **S2** cannot be attributed to one specific assignment. 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. 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 [22]. Likewise, we add two resources to deal with **S2**, one for the total number of assignments in the rotation (r_1) and the other for the number of consecutive assignments to the current shift (r_2).

Finally, the minimum reduced cost rotation of nurse i is obtained through the solution of a variant of the SPPRC where resources are both upper and lower bounded with soft bounds. In contrast, 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. Classical label-setting algorithms for SPPRC [17] can be adapted by computing the penalty associated with r_1 before entering the sink node, and that associated with r_2 at every change of shift type in the rotation. However, the presence of lower bounds and the lack of hard constraints on resources essentially nullifies the efficiency of the dominance rule used in labelling algorithms if no specific method is used. In Section 5.3, we describe how the classical network presented in this section can be modified to solve the pricing problem as a standard SPPRC with only one resource that counts the total number of assignment.

4.3. Branching rules

In this section, we describe the two branching rules implemented in our branch-and-price algorithm. The implementation choices related to the choice of the branching rule and the exploration of the branching tree are detailed in Section 5.4.

In the description of the branching rules, the arcs and flow all refer to those of the rostering graph of a given nurse i (Figure 1). We will say that a rotation/rest arc *covers* a day k when it corresponds to a rotation/resting period that includes day k.

4.3.1. Branching on days

The first rule consists in branching when the sum of the flows on the rest arcs covering any day k is not integer. We then create two branches to ensure that this value is integer. In the first one (the *work branch*),

the nurse is compelled to work on day k, and in the other one (the *rest branch*), he/she is compelled to rest. This is ensured by deleting the rest arcs covering k in the work branch and by deleting the rotation arcs covering k in the rest branch (respectively). What is more, every arc ending at or starting from day k are deleted from the pricing problem of nurse i in the rest branch, so that no rotation arc covering k is generated. In contrast, the pricing remains unchanged in the work branch.

4.3.2. Branching on assignments

The first branching rule does not guarantee the exhaustive enumeration of the feasible schedules, so we implement a branching rule, similar to the previous one, that affects assignments. If the sum of the flows on the rotation arcs including some assignment p is not integer, we create two branches where we either enforce that the nurse has p in his/her roster or not. The consequences on the rostering graph and pricing are similar to those described in the first rule.

5. Implementation choices in our branch-and-price procedure

In this section, we complete the general presentation of our branch-and-price algorithm with the choices we made in our implementation. In particular, we show that the problem can be decomposed according to subsets of independent nurses (Section 5.1) and that groups of nurses can be aggregated in the skill assignment constraints (Section 5.2). In Section 5.3, we describe how we modify the pricing network so that the pricing problem can be solved as a standard SPPRC. Finally, we detail our branching strategies in Section 5.4 and develop a primal heuristic in Section 5.5.

5.1. Decomposition into independent groups of nurses

The nurses are only linked together by the staffing constraints **H1** and **S1**. Both provide lower bounds on the number of nurses assigned to a given shift with a given skill. As a consequence, if several sets of nurses do not share any skill, the problem can be solved independently on each of these sets.

For an optimal separation of the nurses, we build an undirected *skill graph*, $< \mathcal{V}_\Sigma, \mathcal{E}_\Sigma >$, whose vertices correspond to the skills of Σ, and whose edges join two skills if they are both available to at least one nurse, i.e.,

$$\mathcal{E}_\Sigma = \{\{\sigma_1, \sigma_2\} : \exists i \in \mathcal{N} \text{ such that } i \in \mathcal{N}_{\sigma_1} \cap \mathcal{N}_{\sigma_2}\}.$$

The partition of $< \mathcal{V}_\Sigma, \mathcal{E}_\Sigma >$ into its P connected components, $< \mathcal{V}_{\Sigma_1}, \mathcal{E}_{\Sigma_1} >, \ldots, < \mathcal{V}_{\Sigma_P}, \mathcal{E}_{\Sigma_P} >$, provides the maximal subsets of independent skills. Now, we can also partition the set of nurses into P independent subsets, $\mathcal{N}_1, \ldots, \mathcal{N}_P$, where a nurse i is in $\mathcal{N}_p$ if all his/her skills are in Σ_p.

In the instances of INRC–II, this operation leads to two independent problems where the largest one includes about 75% nurses: the smaller one includes the trainee nurses (who have only the skill trainee) and the other one gathers all the other nurses (e.g., head nurses, nurses and caretakers) who share common skills (e.g., a head nurse can work as a head nurse or a nurse, a nurse can work as a nurse or a caretaker, and a caretaker can work only as a caretaker).

5.2. Aggregation of nurses in skill assignment constraints

In a team where the number of skills is small when compared to the number of nurses, it is expected that several nurses share the exact same set of skills. If such nurses work on the same shift, they are equivalent from the perspective of skill assignment. As a consequence, we aggregate the constraints (1e) for the nurses sharing the same set of skills. The resulting modifications in the IP formulation are straightforward, so we do not detail them.

In the instances of INRC–II, we observed that only 4 to 5 groups of similar nurses need to be considered when assigning the skills, instead of 30 to 120 individual nurses.

5.3. Modifications of the pricing network for consecutive assignments constraints

As stated in Section 4.2, the pricing problem can be solved by adapting classical label-setting algorithms to take into account upper and lower soft bounds in the SPPRC. Overall, the algorithmic challenge is twofold. First, the SPPRC label-setting algorithms rely on hard constraints to remove labels that exceed the bounds. However, soft constraints mean that the values of the resources can take higher values than the upper bounds and that the labels cannot be removed. Second, label-setting algorithms rely on dominance rules to converge quickly. Since the consecutive assignments are both upper and lower bounded, one can only conclude on the dominance of one path towards another based on the value of a resource in two situations: when this value is the same for both paths and when the value is greater than the lower bound. Indeed, in the other situations, 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.

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.* [13] include soft time windows to schedule deliveries in a network where the paths are already fixed. Braekers and Janssens [5] modify a label-setting algorithm to apply dominance rules with soft time windows, and they apply their algorithm to small instances. Qurashi *et al.* [29] 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 [30]. They fail in solving vehicle routing problems with more than 7 vehicles and 25 customers.

In our implementation, we modify the pricing network to reduce the pricing problem to a standard SPPRC in a larger graph. First, the penalties on the minimum number of consecutive assignments are handled by enumerating short rotations. Second, the penalties on the number of consecutive assignments to the same shift are managed by adding network layers and arcs. One important motivation for these modifications is that they allows to use efficient implementations of SPPRC label-setting algorithms. What is more, the enumeration of the resulting number of nodes and arcs improves the asymptotic worst-case complexity when the planning horizon grows (see Section 5.3.3).

In the description of the modified pricing network below, we focus on the sub-network associated with one given nurse.

5.3.1. Consecutive assignments to the same shift

The penalty for N consecutive assignments to a given shift s is

$$\begin{cases} c_{\mathsf{S2b}}\left(\mathrm{CS}_s^- - N\right) & \text{if } N < \mathrm{CS}_s^- \\ 0 & \text{if } \mathrm{CS}_s^- \leq N \leq \mathrm{CS}_s^+ \\ c_{\mathsf{S2b}}\left(N - \mathrm{CS}_s^+\right) & \text{if } N > \mathrm{CS}_s^+ \end{cases}$$

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 assignment node $(A_{k,s}, \forall(k,s))$ CS_s^+ times; the duplicates are denoted as $A_{k,s}^n$, $n = 1, \ldots, \mathrm{CS}_s^+$. A path going through node $A_{k,s}^n$ will then correspond to a rotation in which (k,s) is assigned and is either the n-th consecutive assignment to s, or *at least* the n-th one when $n = \mathrm{CS}_s^+$.

For each type of shift s, the assignment nodes form a block $\mathcal{A}_s$. We draw an example focusing on a specific block in Figure 3. The arcs entering the block $\mathcal{A}_s$ denote the beginning of a sequence of assignments to 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 $(A_{k,s}^n, A_{k+1,s}^{n+1})$, and the subsequent assignments correspond to the dotted horizontal arcs $(A_{k,s}^{\mathrm{CS}_s^+}, A_{k+1,s}^{\mathrm{CS}_s^+})$. A cost c_{S2b} is then added to the horizontal arcs because they are used only when more than CS_s^+ consecutive assignments to the shift have occurred, whereas the costs of the plain and diagonal arcs take no penalty due to **S2**. Notice that the arcs exiting from the block only leave from vertices $A_{k,s}^{\mathrm{CS}_s^+}, k \in \{1, \ldots, K\}$. 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 ($\leq \mathrm{CS}_s^-$).

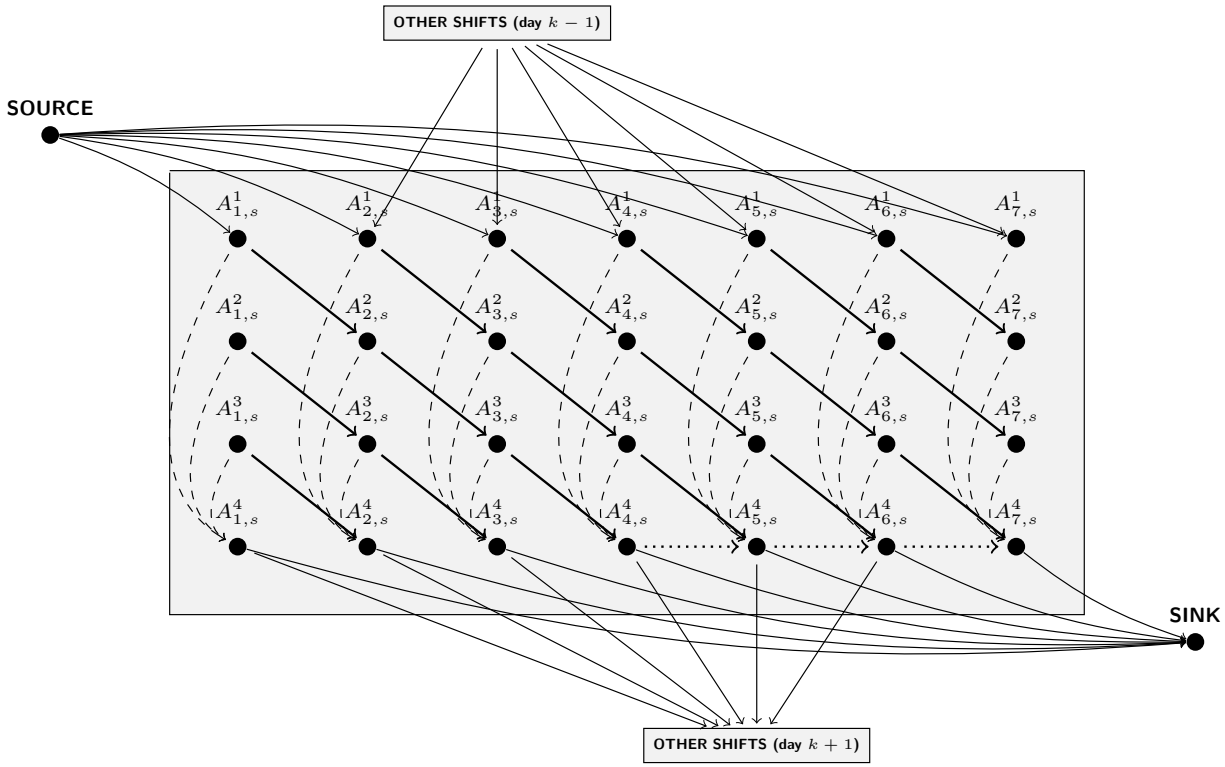


Figure 3: Example of a pricing network of a given nurse for a single shift s. Parameters: $K = 7$, $\mathrm{CS}_s^+ = 4$, and $\mathrm{CD}_i^- = 2$.

5.3.2. Consecutive worked days

For nurse $i \in \mathcal{N}$, the consecutive-assignments penalty associated with a rotation of length L is

$$\begin{cases} c_{\mathsf{S2a}}\left(\mathrm{CD}_i^- - L\right) & \text{if } L < \mathrm{CD}_i^- \\ 0 & \text{if } \mathrm{CD}_i^- \leq L \leq \mathrm{CD}_i^+ \\ c_{\mathsf{S2a}}\left(L - \mathrm{CD}_i^+\right) & \text{if } L > \mathrm{CD}_i^+ \end{cases}$$

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^- assignments. 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.

To handle the upper bound, we then add a resource that counts 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 3 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.

5.3.3. Assessment of the algorithmic complexity

Based on the above description of the modified pricing network, $< \bar{\mathcal{V}}_i, \bar{\mathcal{A}}_i >$, of a given nurse, it is possible to enumerate its arcs to get:

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

What is more, the resulting SPPRC has only one upper bounded resource that counts the length of the rotation. Since the bound on the resource is soft, a rotation can be as long as the planning horizon, hence a label-setting will have a complexity in

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

In comparison, the initial pricing network contains $\mathcal{O}(K \times |\mathcal{S}|^2)$ and the corresponding SPPRC has two resources bounded by K. Hence, the label-setting algorithm has complexity in $\mathcal{O}(K^3 \times |\mathcal{S}|^2)$.

Since the parameters CD_i^-, CS_s^+ and $|\mathcal{S}|$ do not necessarily grow with the size of the planning horizon, we can deduce that the modified pricing network yields a better asymptotic complexity when the planning horizon grows. More importantly, the removal of the lower bounds is likely to yield more dominance in the execution of the label-setting algorithm.

5.4. Branching strategies

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 unexplored 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 (see Section 4.3.1) over branching on shifts (see Section 4.3.2).
- In both branching rules, we branch on the most fractional values. In this selection, we give a small advantage to week-end days because they are involved in more soft constraints and therefore have more influence on the objective value.
- The order in which the two children branching nodes are inserted is random.

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

To find an integer solution early in the branch-and-price and increase the chance of pruning nodes of the branching tree, we use a primal *diving* heuristic. The heuristic is similar to the *pure diving algorithm* described by Sadykov *et al.* [33] who reported fast computation of good primal bounds on several different benchmarks. The heuristic starts with a solution of the linear relaxation of the master problem, $\mathbf{x}^{LR}$. It then sequentially fixes a set of fractional variables to the closest integers and re-optimizes the problem until an integer solution is found. More precisely, our heuristic performs the following steps:

(i) Let $C > 0$ be a threshold expressing the aggressivity of the diving heuristic.

(ii) Choose a set of rotations $\Theta = \bigcup_{i\in\mathcal{N}} \Theta_i$ such that they do not overlap (i.e., contain no assignment on the same day) and are all separated by at least one day off, and

$$\sum_{i\in\mathcal{N}} \sum_{j\in\Theta_i} (1 - x_{ij}^{LR}) \leq C. \tag{3}$$

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

(iv) 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 (ii).

The above algorithm is an adaptation of the pure diving heuristic, where more than one fractional variable can be fixed at each iteration. In particular, we require fixed rotations to be separated by at least one day-off to reduce the risk that the problem becomes infeasible. The parameter C allows to dive more or less

aggressively in the enumeration tree. In our tests, we typically set C to half the number of weeks in the planning horizon.

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

6. An adaptive large neighborhood search for large instances

In preliminary tests, we observed that the branch-and-price algorithm described in previous sections finds the optimal solution only for the smallest 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 ALNS procedure based on the solution of smaller IPs, or equivalently, on the successive fixing and release of some of the variables (see [27] for a recent review on ALNS).

An ALNS algorithm is an iterative process that destroys a part of the current solution at each iteration and repairs 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 the repair phase, a new solution is built by finding a feasible solution for the variables that have been freed. 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. The resulting problem can then be solved to optimality if the number of free variables is sufficiently small. Typically, for the NSP, one can destroy the rosters of some nurses (freeing the corresponding rotations). Then, without changing the other nurses' rosters, the destroyed rosters can be re-optimized by branch-and-price. 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 values 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.

In Section 6.1, 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 6.2. In Section 6.3, we describe a rolling-horizon procedure that we use to determine the initial solution fed to the ALNS.

6.1. Destruction operators

For the process of destructing the current solution, we propose two main strategies: destroy the complete rosters of a small set of nurses, or partially destroy the rosters of a larger set of nurses over a restricted time period. For both strategies, the nurses whose rosters are partially or completely destroyed are called *free nurses*, and the others are called *fixed nurses*. In terms of implementation, if the roster of a free nurse is completely destroyed, the variables associated with his/her rotations, rest periods and skill assignments are set free. In contrast, the variables associated with the rosters of the fixed nurses are fixed to their current value.

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 master problem variables. 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 pricing network by deleting the starting arcs corresponding to rotations that start before k_1 or after k_2.

For a unified presentation of the destruction operators, 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 7.2.3.

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. The type of a nurse is defined by the set of skills he/she can perform.

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 L_i^-, L_i^+, CD_i^-, CD_i^+, CR_i^-, CR_i^+ and WE_i^+.

6.2. 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 [28] 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 has 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.

Once the destruction operator is chosen, the nurses are randomly selected according to probabilities favoring balance in the distribution of workload. A bias is added to increase the likelihood of drawing nurses whose numbers of assignments are not within a small margin from the average number of assignments per nurse. The purpose for introducing this bias is to decrease the penalties paid for violations of the total assignments constraints (**S6**) .

6.3. 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 (whether unknown, noisy or depending on external forces) [31]. 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 4). 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 estimates the impact of the control horizon schedule 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 [32] 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 and we can say that there is no external dynamics, hence the algorithm should perform efficiently.

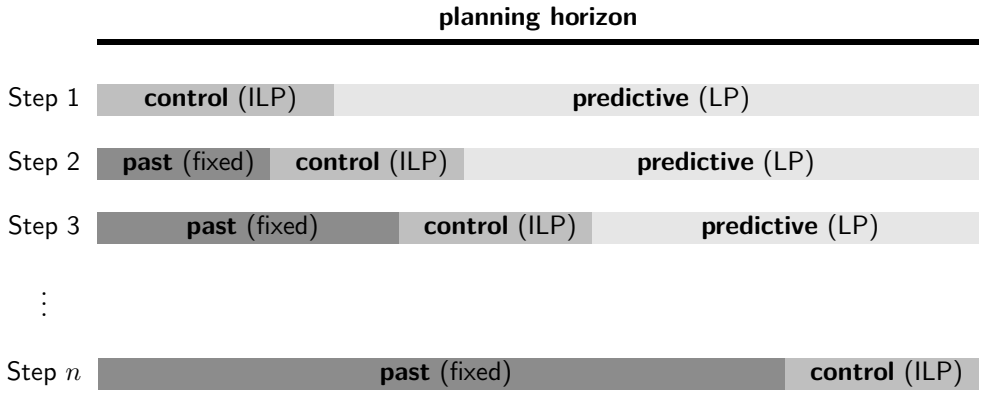


Figure 4: Rolling-horizon procedure. Variables corresponding to rotations starting in the past, control, and prediction horizon are respectively fixed, integer, and relaxed (i.e., may be fractional).

7. Numerical tests

7.1. Instances and benchmark

Instances. We test our algorithm on the instances of the INRC–II competition[3]. The size of the service ranges from 30 to 120 nurses that can perform up to four different skills. The length of the planing horizon is either four or eight weeks and each day is divided in up to four shifts.

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 [10]). All the instances are listed in Table 6 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 $(\mathrm{UB}^* - \mathrm{LB}^*)/\mathrm{LB}^*$.

7.2. 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[4], and the pricing problems are solved with the resource constrained shortest path from the Boost library[5]. 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 [25] under an MIT licence.

The figures reproduced in this section represent the distributions 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

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

[4]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/`

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

$\lvert\mathcal{N}\rvert$	$M = 4$				$M = 8$			
	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 6: Best results obtained on the benchmark.

the top (bottom) of the box (see [14] for a more detailed description). Finally, we defined these following measures. $\text{Gap}^0 = (\text{UB}^0 - \text{LB}^*)/\text{LB}^*$ designates the gap of the initial solution, where UB^0 is the best upper bound at the end of the initialization. Similarly, $\text{Gap} = (\text{UB} - \text{LB}^*)/\text{LB}^*$ designates the gap of the best solution obtained after the ALNS has been run.

In preliminary tests, we observed that some initialization methods (which finally proved to be the best ones in our experimental settings) could exceed the time limit set by the INRC–II. As we aim at evaluating different initialization strategies and different ALNS settings, it is necessary to have some time left after the initialization, and so we set a higher time limit. We also use a formula that depends on the number of nurses and weeks to compute the time limit, but we settled for $M \times [6\,|\mathcal{N}|] + 60$ seconds instead of the approximate $M \times [10 + 3\,(|\mathcal{N}| - 20)]$ seconds suggested in the INRC–II (M is the number of weeks in the planning horizon). Note that the INRC–II time limit is relatively small but consistent with a dynamic process where a one-week schedule is computed at each step; however, it makes sense to increase this limit in a static context, as usually more computational time is available.

Besides the 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 lower bounds reported in Table 6 (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.

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

In Figure 5, 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 diving heuristic (see Section 5.5) provide no improvement in the upper bound. The sampling horizon is equal to one week in all our tests.

First, from Figures 5a and 5b, 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 5c and 5d, 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 5e and 5f show, as expected, that the longer the control horizon, the more time is spent in the initialization.

7.2.2. Performance of the initialization method

In Figure 6, 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 diving heuristic has been run twice, and after two successive executions of this heuristic without 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 6a–6d, 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 6e and 6f, 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, a good initialization is worth the larger time spent.

7.2.3. Influence of the destruction operator on the ALNS

In this Section, we compare the destruction operators of the ALNS that are presented in Section 6.1. The results are displayed in Figures 7a–7d, where we focus on the best upper bound found by the solution algorithm. We note that in the repair step of the ALNS, the branch-and-price is run until optimality or until two successive calls to the diving heuristic without improvement. For most destruction operators though, the repair step problem is sufficiently small to be solved to optimality before the diving heuristic fails twice in a row.

On Figures 7a–7b, 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 7c–7d, we report the comparison of the following ALNS strategies (label names are specified in parentheses):

- only complete rosters are destroyed ("complete"),
- only partial rosters are destroyed ("partial"),
- selected nurses always have the same type ("`D_Type`"),
- selected nurses always have the same contract ("`D_Contract`"),
- selected nurses are always picked randomly among all the nurses ("`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 the integral gap of the ALNS where every destruction operator is used in the roulette-wheel 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 rosters 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.

7.2.4. Comparison with other methods

To conclude the tests, we compare the results of our method with those found in the literature for the same benchmark. The experimental comparison is summarized in Table 7. In the "BKS" column, we display the value of the best known solution (including the upper bounds found during our tests). For each other method, we display the computational time limit ("cpu" column) and the gaps to BKS ("default", "best" and "average" columns). The results found under the "ALNS" columns are produced by our method. As a result of the sensitivity tests discussed above, our default settings are to search for an initial solution by rolling horizon and destroy $N_N \times N_W = 48$ weeks at each iteration of the ALNS. In column "default", we report the gap to BKS for one execution with default settings and in column "best", we report the best gap found during our tests. In the "dynamic" section, we report the best results found during the INRC–II competition [8]. 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. The values indicated for the dynamic problem are the best results over the 15 participating teams and all the random seeds input by the organizers. In the section [9] of the table, we display the results produced by Ceschia and Schaerf [9] with a simulated annealing algorithm using large neighborhoods, where the schedules of nurses are changed or swapped over k consecutive days. They report the best and average upper bound (columns "best" and "avg.") over 30 repetitions with different random seeds. Gomes, Toffolo and Santos [19] developed a branch-and-price matheuristic. Their column generation scheme is based on a decomposition by roster (each column is a schedule on the complete horizon), and they develop a variable neighborhood search (VNS) for fast computation of upper bounds. Finally, the results reported in [35] are for a matheuristic where a VNS is used in conjunction with a compact integer programming formulation. In the last two references, the authors report the results they obtained with their default settings, but only for a subset of the 4-weeks instances.

Note that the organizers of INRC–II provided a tool that estimates time limits depending on the performance of the computer. The time limits reported for the dynamic problem, ALNS and [9] are all set with this tool and these tests were executed on a single thread, so the values in "cpu" columns are comparable. In contrast, [19] used an Intel(R) Xeon E5620@2.40GHz 8-Core processor with parallel computing, and in [35] the tests are run on a single thread of an Intel Core i5-2410M CPU@2.30GHz processor.

First, we observe that our method was able to compute the BKS for every instance except four (which are on a 4-weeks horizon). These are mostly obtained with other settings than default, but the results found with default settings are still consistently close to the best ones.

It is comforting to observe that default ALNS achieves a significant improvement with respect to the best solutions of the dynamic problem (2.3% average gap instead of 16.1%). The largest improvements are obtained for the 8 weeks instances, which is 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), default ALNS does not 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, our results with default settings reflect only one specific execution of the algorithm.

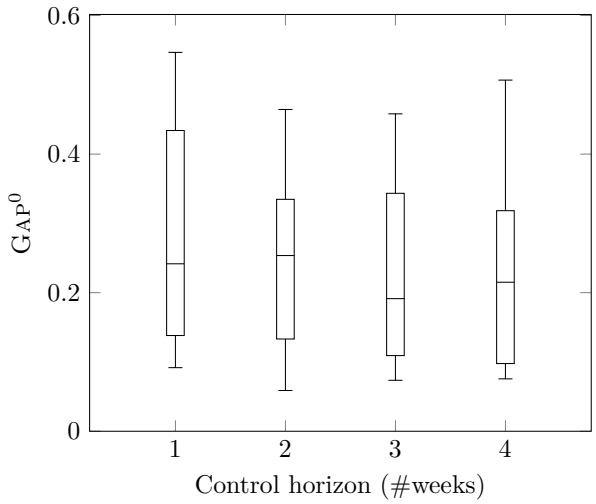
The comparison with [9] seems to confirm the relevance of a branch-and-price matheuristic when compared to a pure metaheuristics. Our opinion is that, in this context, branch-and-price allows for efficient generation of improving rotations, which can be more challenging with heuristic moves. The benefit of column generation is particularly important for the 8 weeks horizon.

Finally, the most significant difference of our method with that developed by Gomes *et al.* [19] is in the choice of a rotation-based decomposition. Hence their result allow for a partial assessment of this choice. Even though they used larger time limits and parallel computation on 8 cores, [19] found the BKS of only one small instance (n035w4_0_1-7-1-8) out of six. Overall, with default settings, [19] obtains a 3.9% gap on the six instances, while ALNS reaches a 2.2% gap. This comparison is very promising for rotation-based decomposition, but we still think that there is a need for a more rigorous comparison before drawing

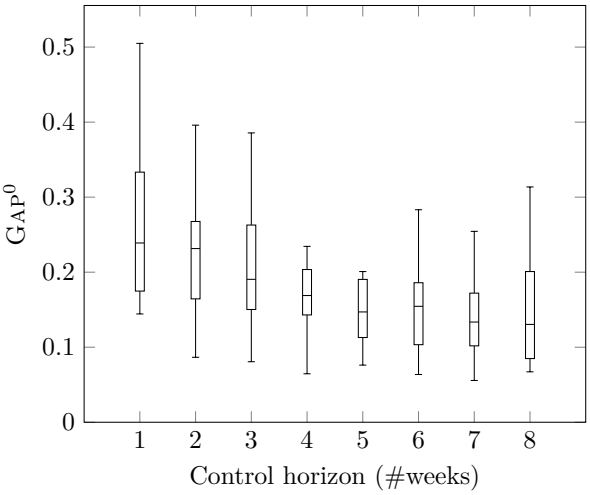
instance	BKS	dynamic		ALNS			[9]			[19]		[35]	
		cpu	best	cpu	default	best	cpu	avg.	best	cpu	default	cpu	default
n030w4_1_6-2-9-1	1685	192	4.2%	780	0.6%	**0.0%**	780	2.7%	0.9%	–	–	–	–
n030w4_1_6-7-5-3	1840	192	5.2%	780	2.7%	**0.0%**	780	2.8%	1.1%	–	–	–	–
n035w4_0_1-7-1-8	1415	264	15.2%	900	0.7%	**0.0%**	900	7.3%	3.2%	3269	0.7%	7200	11.6%
n035w4_2_8-8-7-5	1085	264	15.7%	900	6.5%	5.5%	900	12.0%	6.5%	5586	**0.0%**	7200	11.4%
n040w4_0_2-0-6-1	1640	336	5.5%	1020	2.7%	**0.0%**	1020	4.9%	0.9%	–	–	–	–
n040w4_2_6-1-0-6	1850	336	1.6%	1020	2.2%	0.8%	1020	3.3%	**0.0%**	–	–	–	–
n050w4_0_0-4-8-7	1445	480	3.1%	1260	4.2%	**0.0%**	1260	5.6%	1.4%	–	–	–	–
n050w4_0_7-2-7-2	1405	480	5.3%	1260	6.8%	**0.0%**	1260	7.0%	2.5%	–	–	–	–
n060w4_1_6-1-1-5	2465	624	14.2%	1500	1.6%	**0.0%**	1500	10.0%	7.1%	–	–	–	–
n060w4_1_9-6-3-8	2730	624	8.1%	1500	0.7%	**0.0%**	1500	6.9%	2.6%	–	–	–	–
n070w4_0_3-6-5-1	2430	768	11.1%	1740	0.2%	**0.0%**	1740	5.6%	1.4%	3640	1.2%	7200	22.3%
n070w4_0_4-9-6-7	2125	768	14.4%	1740	2.4%	**0.0%**	1740	8.9%	4.9%	4943	9.6%	7200	26.4%
n080w4_2_4-3-3-3	3320	912	6.5%	1980	0.6%	**0.0%**	1980	7.9%	5.0%	–	–	–	–
n080w4_2_6-0-4-8	3240	912	10.2%	1980	0.6%	**0.0%**	1980	7.2%	4.3%	–	–	–	–
n100w4_0_1-1-0-8	1230	1200	17.5%	2460	1.2%	**0.0%**	2460	20.4%	10.6%	–	–	–	–
n100w4_2_0-6-4-6	1855	1200	13.2%	2460	5.1%	**0.0%**	2460	11.8%	7.0%	–	–	–	–
n110w4_0_1-4-2-8	2390	1344	13.4%	2700	2.1%	**0.0%**	2700	6.7%	2.7%	13084	7.1%	7200	25.4%
n110w4_0_1-9-3-5	2525	1344	15.6%	2700	1.4%	**0.0%**	2700	7.9%	5.1%	9624	4.6%	7200	40.6%
n120w4_1_4-6-2-6	1880	1488	29.5%	2940	15.4%	15.2%	2940	6.0%	**0.0%**	–	–	–	–
n120w4_1_5-6-9-8	2015	1488	23.3%	2940	10.2%	10.2%	2940	3.6%	**0.0%**	–	–	–	–
Average 4 weeks			11.6%		3.4%	1.6%		7.4%	3.4%		–		–
n030w8_1_2-7-0-9-3-6-0-6	2070	384	13.0%	1500	2.7%	**0.0%**	1500	4.7%	1.9%	–	–	–	–
n030w8_1_6-7-5-3-5-6-2-9	1735	384	9.5%	1500	**0.0%**	**0.0%**	1500	7.0%	3.5%	–	–	–	–
n035w8_0_6-2-9-8-7-7-9-8	2555	528	18.2%	1740	0.6%	**0.0%**	1740	13.5%	7.8%	–	–	–	–
n035w8_1_0-8-1-6-1-7-2-0	2305	528	20.2%	1740	1.1%	**0.0%**	1740	16.4%	9.1%	–	–	–	–
n040w8_0_0-6-8-9-2-6-6-4	2620	672	26.3%	1980	0.6%	**0.0%**	1980	13.3%	9.4%	–	–	–	–
n040w8_2_5-0-4-8-7-1-7-2	2420	672	11.6%	1980	3.1%	**0.0%**	1980	14.0%	10.5%	–	–	–	–
n050w8_1_1-7-8-5-7-4-1-8	4900	960	10.4%	2460	1.8%	**0.0%**	2460	6.6%	4.0%	–	–	–	–
n050w8_1_9-7-5-3-8-8-3-1	4925	960	10.4%	2460	1.5%	**0.0%**	2460	4.8%	3.0%	–	–	–	–
n060w8_0_6-2-9-9-0-8-1-3	2345	1248	17.9%	2940	4.1%	**0.0%**	2940	17.1%	11.9%	–	–	–	–
n060w8_2_1-0-3-4-0-3-9-1	2590	1248	18.3%	2940	**0.0%**	**0.0%**	2940	15.2%	10.8%	–	–	–	–
n070w8_0_3-3-9-2-3-7-5-2	4595	1536	11.3%	3420	1.4%	**0.0%**	3420	12.4%	9.6%	–	–	–	–
n070w8_0_9-3-0-7-2-1-1-0	4760	1536	13.2%	3420	0.2%	**0.0%**	3420	10.9%	9.1%	–	–	–	–
n080w8_1_4-4-9-9-3-6-0-5	4180	1824	19.5%	3900	1.2%	**0.0%**	3900	18.2%	14.7%	–	–	–	–
n080w8_2_0-4-0-9-1-9-6-2	4450	1824	13.0%	3900	1.0%	**0.0%**	3900	17.5%	13.5%	–	–	–	–
n100w8_0_0-1-7-8-9-1-5-4	2125	2400	44.9%	4860	0.9%	**0.0%**	4860	35.5%	30.1%	–	–	–	–
n100w8_1_2-4-7-9-3-9-2-8	2210	2400	38.2%	4860	1.8%	**0.0%**	4860	39.7%	32.8%	–	–	–	–
n110w8_0_2-1-1-7-2-6-4-7	4010	2688	28.6%	5340	**0.0%**	**0.0%**	5340	15.5%	11.8%	–	–	–	–
n110w8_0_3-2-4-9-4-1-3-7	3560	2688	35.0%	5340	0.4%	**0.0%**	5340	17.1%	12.8%	–	–	–	–
n120w8_0_0-9-9-4-5-1-0-3	2600	2976	39.0%	5820	0.8%	**0.0%**	5820	30.6%	24.6%	–	–	–	–
n120w8_1_7-2-6-4-5-2-0-2	3095	2976	13.4%	5820	1.0%	**0.0%**	5820	21.6%	16.8%	–	–	–	–
Average 8 weeks			20.6%		1.2%	0.0%		16.6%	12.4%		–		–
Average overall			16.1%		2.3%	0.8%		12.0%	7.9%		–		–

Table 7: Comparison of our method with the literature

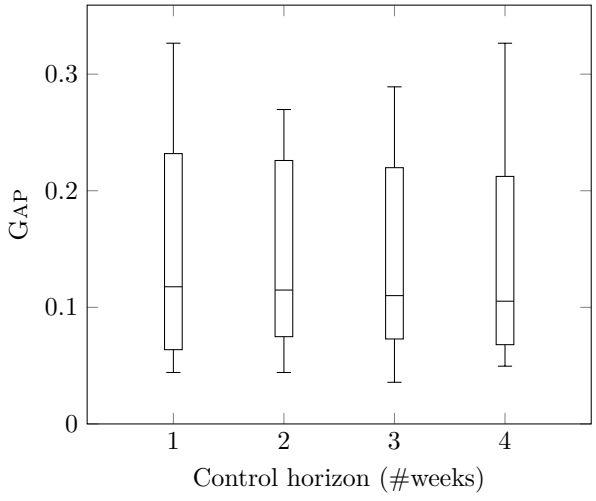
conclusions. In particular, the results reported in [19] seem to be preliminary and they allow only for a comparison on a subset of instances which does not include any 8-weeks instance.



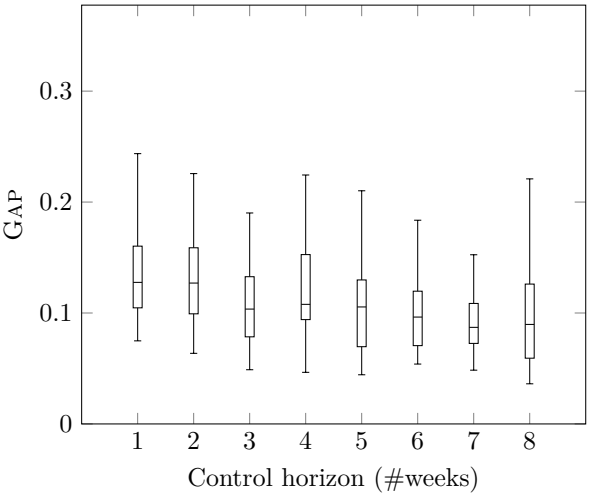
(a) 4-weeks instances.



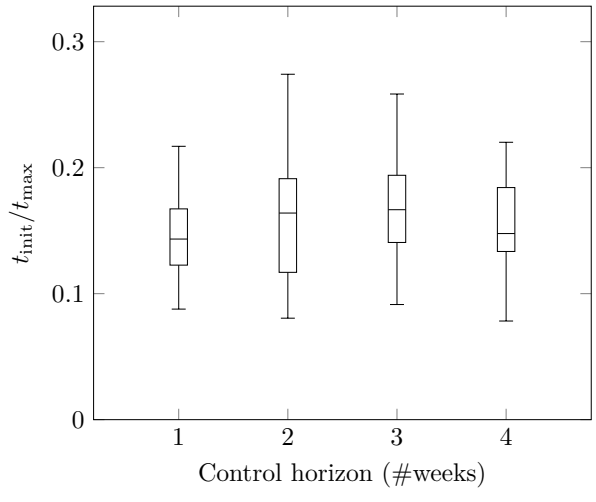
(b) 8-weeks instances.



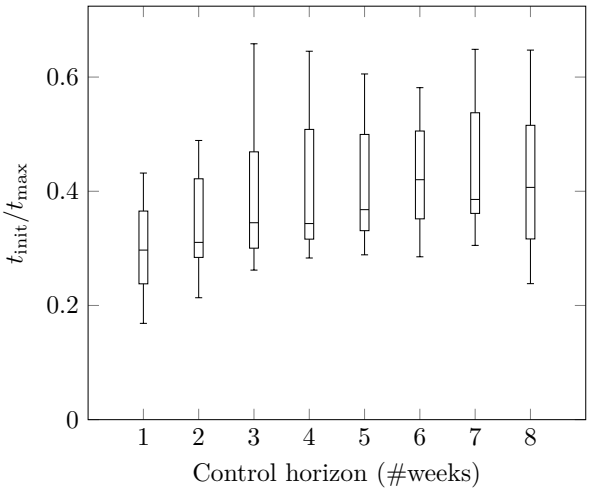
(c) 4-weeks instances.



(d) 8-weeks instances.

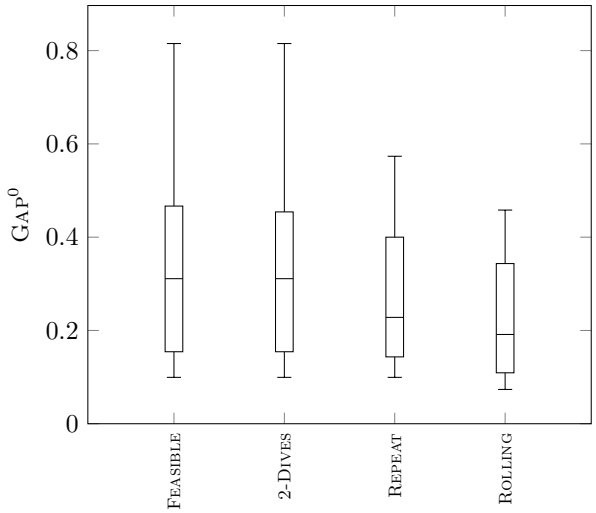


(e) 4-weeks instances.

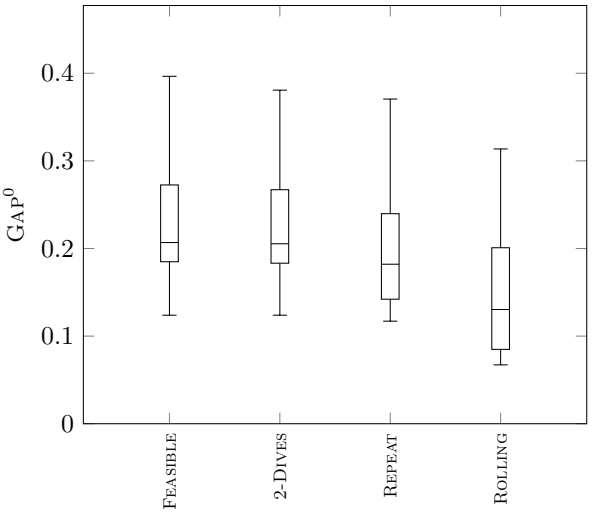


(f) 8-weeks instances.

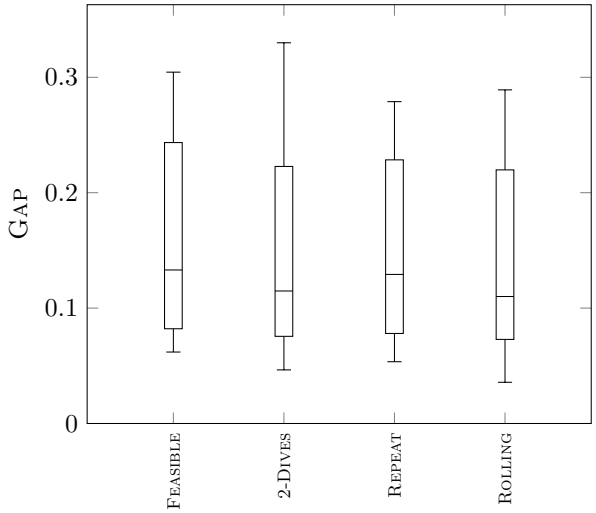
Figure 5: Impact of the size of the control horizon on the rolling-horizon initialization



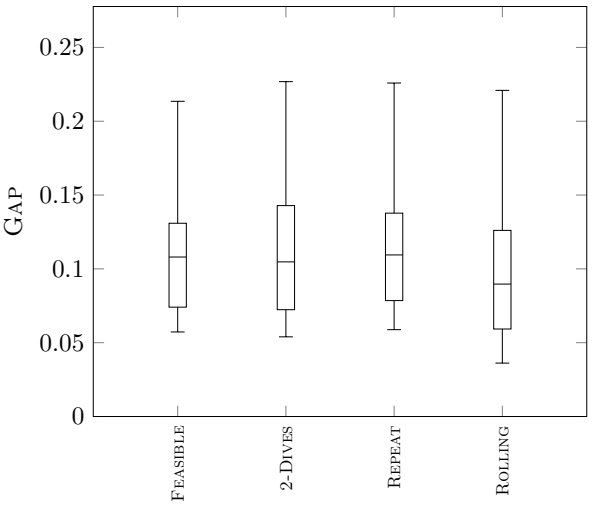
(a) 4-weeks instances.



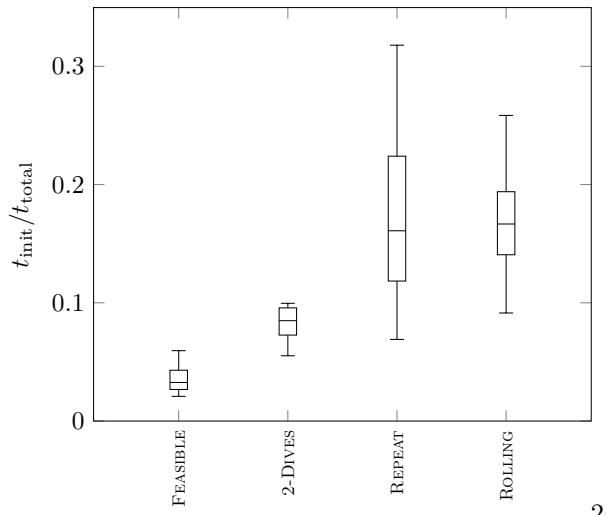
(b) 8-weeks instances.



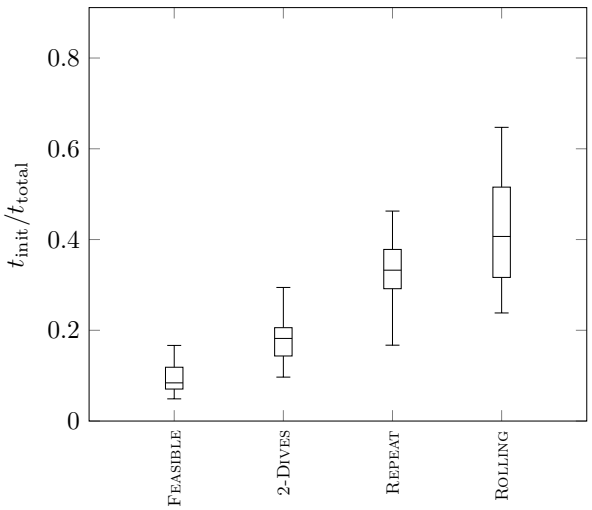
(c) 4-weeks instances.



(d) 8-weeks instances.

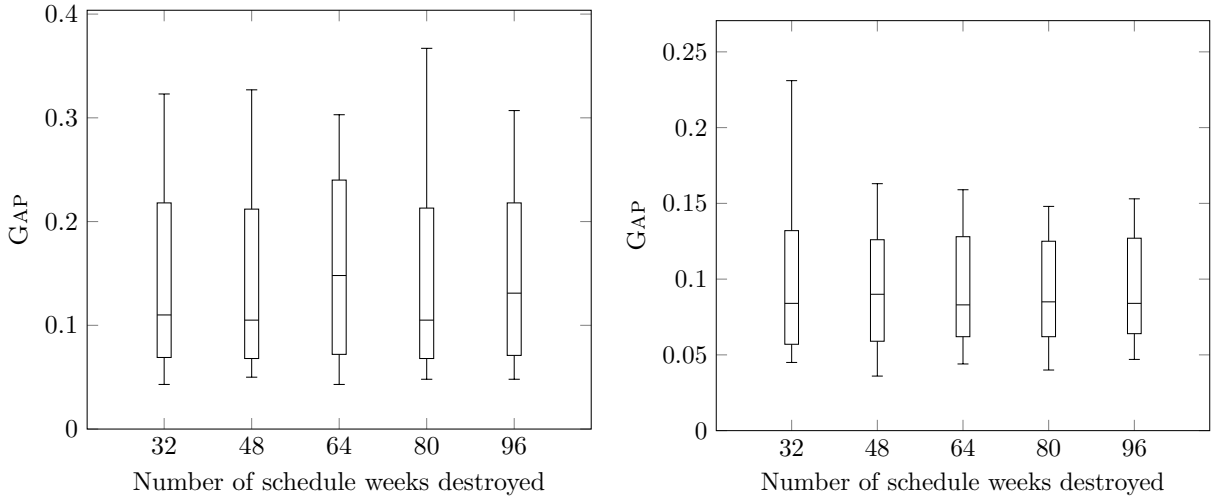


(e) 4-weeks instances.



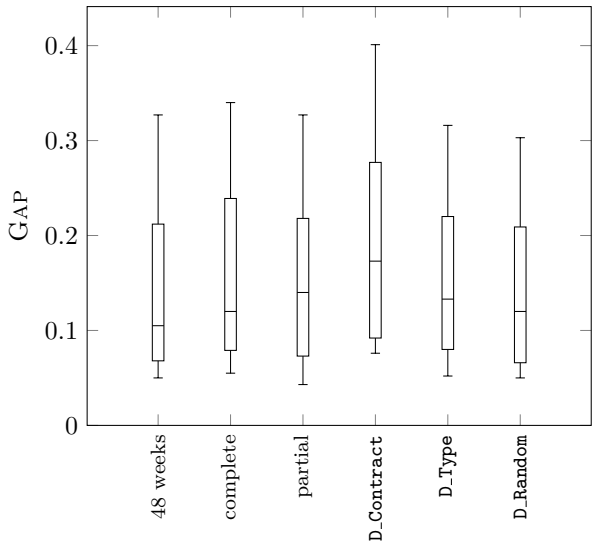
(f) 8-weeks instances.

Figure 6: Comparison of the initialization methods

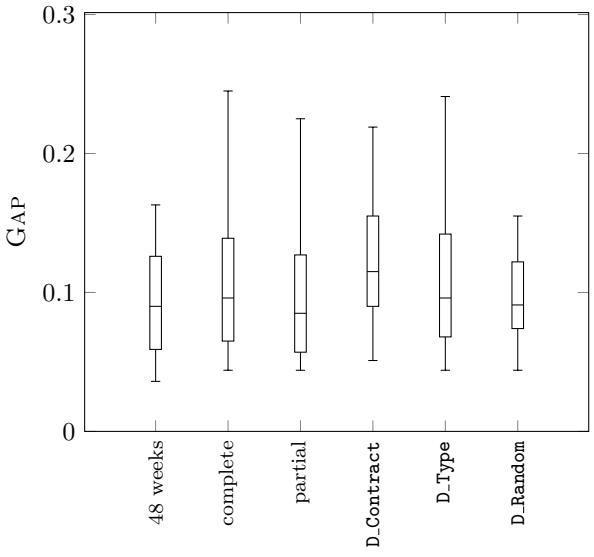


(a) 4-weeks instances.

(b) 8-weeks instances.



(c) 4-weeks instances



(d) 8-weeks instances.

Figure 7: Performance of the ALNS depending on the destruction operator.

8. Conclusions

This article deals with the NSP described in the context of the international competition INRC–II. Our first contribution is the description of a branch-and-price algorithm based on rotations, i.e., sequences of assignments preceded and followed by at least one day off. This decomposition, adapted from aircrew planning, allows to reduce the complexity of the pricing problem and the size of the set of feasible columns. The master problem is a constrained flow problems where the complete schedule is built by concatenation of rotations and resting periods, and skills are chosen for each assignment of the nurses. The pricing problems are modeled as shortest path problems with one resource constraint which corresponds to the total number of assignments in the rotation.

To achieve good results on large instances, we then develop an ALNS behaving as a 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 are concerned with the schedule of 30 to 120 nurses over four and eight 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, the numerical comparison of our algorithm with the literature confirms the relevance of our approach.

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., Purnomo, H.W.: Preference scheduling for nurses using column generation. European Journal of Operational Research **164**(2), 510 – 534 (2005). DOI 10.1016/j.ejor.2003.06.046

[2] Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: Column generation for solving huge integer programs. Operations Research **46**(3), 316–329 (1998). DOI 10.1287/opre.46.3.316

[3] Beliën, J., Demeulemeester, E.: A branch-and-price approach for integrating nurse and surgery scheduling. European Journal of Operational Research **189**(3), 652–668 (2008). DOI 10.1016/j.ejor.2006.10.060

[4] Boyer, V., Gendron, B., Rousseau, L.M.: A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem. Journal of Scheduling **17**(2), 185–197 (2014). DOI 10.1007/s10951-013-0338-9

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

[6] Burke, E.K., Curtois, T.: New approaches to nurse rostering benchmark instances. European Journal of Operational Research **237**(1), 71–81 (2014). DOI 10.1016/j.ejor.2014.01.039

[7] Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. Journal of Scheduling **7**(6), 441–499 (2004). DOI 10.1023/B:JOSH.0000046076.75950.0b

[8] Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: The second international nurse rostering competition. Annals of Operations Research (2018). DOI 10.1007/s10479-018-2816-0

[9] Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: Solving the INRC-II nurse rostering problem by simulated annealing based on large-scale neighborhoods. In: Proceedings of the 12th International Confenference on Practice and Theory of Automated Timetabling (PATAT-2018) (2018)

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

[11] Cheang, B., Li, H., Lim, A., Rodrigues, B.: Nurse rostering problems – a bibliographic survey. European Journal of Operational Research **151**(3), 447–460 (2003)

[12] Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, vol. 5. Springer Science & Business Media (2006)

[13] Dumas, Y., Soumis, F., Desrosiers, J.: Optimizing the schedule for a fixed vehicle path with convex inconvenience costs. Transportation Science **24**(2), 145–152 (1990). DOI 10.1287/trsc.24.2.145

[14] Frigge, M., Hoaglin, D.C., Iglewicz, B.: Some implementations of the boxplot. The American Statistician **43**(1), 50–54 (1989)
[15] Gamache, M., Soumis, F.: A method for optimally solving the rostering problem. In: G. Yu (ed.) Operations Research in the Airline Industry, *International Series in Operations Research and Management Science*, vol. 9, chap. 5, pp. 124–157. Springer US, New York (1998)
[16] Gamache, M., Soumis, F., Marquis, G., Desrosiers, J.: A column generation approach for large-scale aircrew rostering problems. Operations research **47**(2), 247–263 (1999)
[17] Garcia, R.: Resource constrained shortest paths and extensions. Ph.d. thesis, Georgia Institute of Technology, GA, USA (2009)
[18] Gérard, M., Clautiaux, F., Sadykov, R.: Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce. European Journal of Operational Research **252**(3), 1019–1030 (2016). DOI 10.1016/j.ejor.2016.01.036
[19] Gomes, R.A., Toffolo, T.A., Santos, H.G.: Variable neighborhood search accelerated column generation for the nurse rostering problem. Electronic Notes in Discrete Mathematics **58**, 31 – 38 (2017). DOI 10.1016/j.endm.2017.03.005. 4th International Conference on Variable Neighborhood Search
[20] Haspeslagh, S., De Causmaecker, P., Schaerf, A., Stølevik, M.: The first international nurse rostering competition 2010. Annals of Operations Research **218**(1), 221–236 (2014). DOI 10.1007/s10479-012-1062-0
[21] He, F., Qu, R.: A constraint programming based column generation approach to nurse rostering problems. Computers & Operations Research **39**(12), 3331 – 3343 (2012). DOI 10.1016/j.cor.2012.04.018
[22] Irnich, S., Desaulniers, G., et al.: Shortest path problems with resource constraints. In: Column generation, chap. 2, pp. 33–65. Springer US (2005)
[23] Jaumard, B., Semet, F., Vovor, T.: A generalized linear programming model for nurse scheduling. European Journal of Operational Research **107**(1), 1–18 (1998). DOI 10.1016/S0377-2217(97)00330-5
[24] Kohl, N., Karisch, S.E.: Airline crew rostering: Problem types, modeling, and optimization. Annals of Operations Research **127**(1-4), 223–257 (2004)
[25] Legrain, A., Rosat, S., Omer, J.: legraina/nursescheduler: Static rostering (2019). DOI 10.5281/zenodo.3460634. URL `https://doi.org/10.5281/zenodo.3460634`
[26] Maenhout, B., Vanhoucke, M.: Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. Journal of Scheduling **13**(1), 77–93 (2010). DOI 10.1007/s10951-009-0108-x
[27] Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of Metaheuristics, chap. 13, pp. 399–419. Springer US (2010). DOI 10.1007/978-1-4419-1665-5_13
[28] Prescott-Gagnon, E., Desaulniers, G., Rousseau, L.M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks **54**(4), 190–204 (2009). DOI 10.1002/net.20332
[29] Qurashi, A.G., Taniguchi, E., 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)
[30] Qurashi, A.G., Taniguchi, E., Yamada, T.: Exact solution for vehicle routing problem with soft time windows and dynamic travel time. Asian Transport Studies **2**(1), 48–63 (2012)
[31] Richalet, J., Rault, A., Testud, J., Papon, J.: Model predictive heuristic control: applications to industrial processes. Automatica **14**(5), 413 – 428 (1978). DOI 10.1016/0005-1098(78)90001-8
[32] Saddoune, M., Desaulniers, G., Soumis, F.: Aircrew pairings with possible repetitions of the same flight number. Computers & Operations Research **40**(3), 805 – 814 (2013). DOI 10.1287/opre.47.2.247
[33] Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E.: Primal Heuristics for Branch-and-Price: the assets of diving methods. INFORMS Journal on Computing (2018)
[34] Santos, H.G., Toffolo, T.A., Gomes, R.A., Ribas, S.: Integer programming techniques for the nurse rostering problem. Annals of Operations Research **239**(1), 225–251 (2016)
[35] Wickert, T.I., Santori, C.S., Buriol, L.S.: A fix-and-optimize VNS algorithm applied to the nurse rostering problem. In: Proceedings of the Sixth International Workshop on Model-based Metaheuristic (Matheuristics-2016), pp. 1–12 (2016)