Relaxation de contraintes globales pour la modélisation de problème d'allocation d'infirmières

Jean-Philippe Metivier, Patrice Boizumault, Samir Loudni

Relaxation de contraintes globales pour la résolution de nurse-rostering problems

Jean-Philippe Métivier **Patrice Boizumault** **Samir Loudni**

GREYC (CNRS - UMR 6072) – Université de Caen
Boulevard du Maréchal Juin
14000 Caen
jmetivie@info.unicaen.fr {patrice.boizumault,samir.loudni}@unicaen.fr

Résumé

Les *nurse-rostering problems* consistent à affecter, pour chaque infirmière, une équipe sur une période de temps fixée en respectant des règles spécifiant les besoins de l'hôpital et la législation du travail. Ces problèmes sont généralement difficiles à résoudre car de grande taille et surcontraints. L'objectif de cet article est de montrer comment ces problèmes peuvent être modélisés et résolus à l'aide de contraintes globales relaxées dans des temps comparables aux méthodes ad'hoc utilisées.

1 Introduction

Les problèmes d'affectation d'infirmières (NRPs, *nurse-rostering problems*) consistent à affecter, pour chaque infirmière, une équipe (matin, soir, nuit, repos, ...) sur une période de temps fixée (horizon) en respectant des règles spécifiant les besoins de l'hôpital et la législation du travail. De tels problèmes sont généralement difficiles à résoudre car de grande taille et surcontraints. Le site ASAP[1] de l'Université de Nottingham [4] recense un ensemble de NRPs réels, ainsi que les méthodes utilisées jusque-là pour les résoudre. Ces problèmes sont souvent résolus par des méthodes de Recherche Opérationnelle (RO) ad'hoc, la plupart incluant des étapes de pré-taitement.

Dans le cadre d'une approche CSP, les contraintes globales procurent une modélisation élégante et un filtrage très performant pour la résolution des NRPs. Mais, ces problèmes sont par nature surcontraints : des coûts de violation sont associés à la non-satisfaction de certaines règles. L'objectif de cet article est de montrer comment ces problèmes peuvent être modélisés et résolus à l'aide de contraintes globales relaxées dans des temps comparables aux méthodes de RO ad'hoc utilisées.

De manière générale, on peut distinguer deux grandes sortes de règles : celles qui imposent des bornes sur le nombre d'infirmières, d'équipes,... qui seront traduites par des contraintes `Gcc` [15] (ou une de ses versions relaxées Σ-`Gcc` [10]) et celles qui imposent, pour chaque infirmière, des contraintes sur l'enchaînement des équipes qui seront traduites par des contraintes `Regular` [13] (ou une de ses versions relaxées `costRegular` [2]).

La Section 2 fait un panorama synthétique des NRPs et présente un exemple introductif que nous estimons être représentatif des différents NRPs. Dans la Section 3, nous présentons la/les relaxation(s) des contraintes globales `Gcc` et `Regular`. À l'aide de cet exemple, nous montrons (Section 4) comment les NRPs peuvent être modélisés de manière concise et élégante à l'aide des contraintes globales relaxées.

Les 2 sections suivantes sont consacrées à la résolution des NRPs ainsi modélisés. Tout d'abord, nous définissons (Section 5) la contrainte globale `RegularCount`, et sa version relaxée, qui combine les contraintes `Regular` et `Atleast`/`Atmost` afin d'obtenir un filtrage nettement plus performant que celui réalisé séparément par chacune de ces contraintes. Dans la Section 6, nous décrivons la méthode retenue pour nos expérimentations. Il s'agit d'une méthode de type VNS (Variable Neigbourhood Search) où la reconstruction des solutions s'effectue à l'aide d'un LDS (Limited Discrepancy Search) combiné avec le filtrage précédemment décrit.

Dans la Section 7, nous comparons expérimentale-

[1] Automated Scheduling, Optimisation and Planning, http ://www.cs.nott.ac.uk/˜tec/NRP/

ment notre approche avec les méthodes de RO utilisées sur un ensemble de problèmes réels extraits du site du ASAP. Nos expérimentations montrent que, malgré la flexibilité et la généricité de notre approche, des solutions de bonnes qualités peuvent être obtenues rapidement et que cette approche est compétitive par rapport aux méthodes ad'hoc développées jusque là.

2 Problème d'affection d'infirmières

2.1 Description du problème

Un NRP consiste à affecter, pour chaque infirmière, une équipe (*shift*) sur une période de temps fixée (*planning horizon*) en respectant des règles spécifiant les besoins de l'hôpital et la législation du travail. Le but est de trouver un affectation complète satisfaisant au mieux les contraintes du problème.

À chaque équipe est associée une période de temps durant laquelle une infirmière est en service. On distingue généralement les équipes suivantes : l'équipe du matin M, du soir E et celle de nuit N. Pour représenter les périodes de repos, on introduit l'équipe de repos R. On notera par *2-shifts problem*, les problèmes constitués des équipes M et N de durée égale à 12 heures, et par *3-shifts problem*, ceux constitués des équipes M, E et N de durée de 8 heures.

Chaque équipe doit respecter des exigences en terme de nombre d'infirmières requis et de qualification de celles-ci (contraintes d'équipes "*shift constraints*"). Par ailleurs, le planning d'une infirmière doit tenir compte de règles sur les séquences de travail (longueur maximale de ces enchaînements, type d'enchaînement,...) et des préférences des infirmières (contraintes d'infirmières "*nurse constraints*"). Notons qu'une infirmière ne doit pas travailler dans plus d'une équipe par jour. Parmi ces exigences, certaines doivent être absolement respectées (contraintes d'intégrité) alors que d'autres peuvent ne pas être satisfaites (contraintes de préférences dont les coûts de violation sont donnés dans la spécification même du problème).

2.2 Exemple introductif

Considérons le problème à *3-shifts* constitué de 8 infirmières travaillant sur un horizon de 21 jours et devant respecter les règles suivantes[2].

- Contraintes d'intégrité
  - (I1) Les équipes du matin M et du soir E sont composées de 2 infirmières et celle de nuit N d'une infirmière ;
  - (I2) Une infirmière ne doit pas travailler plus de quatre fois en équipe de nuit.
  - (I3) Sur une période de 21 jours, une infirmière doit au moins avoir sept repos.
  - (I4) Une infirmière doit avoir au moins un dimanche de libre toutes les trois semaines.
  - (I5) Une infirmière doit au plus travailler trois nuits consécutives.
  - (I6) Deux équipes successives doivent être séparées d'au moins 8 heures de repos.
- Contraintes de préférences
  - (P1) Pour une infirmière, un repos isolé est pénalisé par un coût de 100.
  - (P2) Une infirmière doit travailler de 4 à 8 fois en équipe du matin et de 4 à 8 fois dans celle du soir. Tout manque ou excès (δ) est pénalisé par un coût de ($10\times\delta$).
  - (P3) Une infirmière ne doit pas travailler plus de 4 jours consécutifs. Chaque jour supplémentaire entraine une pénalité de 1000.
  - (P4) Chaque nuit isolée entraîne une pénalité de 100.
  - (P5) Deux équipes successives doivent être séparées d'au moins 16 heures de repos. Tout écart inférieur est pénalisé d'un coût de 100.

La régle (I1) est une *shift constraint* alors que les autres sont des *nurse constraints*.

3 Relaxation de contraintes globales

3.1 Relaxer une contrainte globale

Relaxer une contrainte globale [17] c'est :

- **définir une sémantique de violation** permettant de quantifier le degré de violation de la contrainte globale ;
- **déterminer le niveau de cohérence** que l'on souhaite maintenir ;
- **proposer un test de cohérence et un algorithme de filtrage** associé.

Il peut exister plusieurs relaxations d'une même contrainte globale selon la sémantique de violation considérée. Le niveau de cohérence souhaité est bien entendu global, mais il peut être plus faible en l'absence d'algorithmes de test de cohérence et de filtrage performants. Par exemple, le test de cohérence est un problème NP-Complet pour la relaxation de la contrainte globale `AllDifferent` pour la sémantique de violation basée décomposition [14, 10], alors que ce même test est de complexité polynomiale faible pour la relaxation de `AllDifferent` pour la sémantique de violation basée variables [9].

[2] Ces règles, extraites de problèmes réels (cf. [4]), nous semblent représentatives des NRP.

3.2 Relaxer la contrainte globale Gcc

3.2.1 Global Cardinality Constraint

La contrainte globale `Gcc` [15] impose qu'un ensemble de variables prenne ses valeurs de façon à respecter des bornes inférieures et supérieures sur le nombre de fois que ces valeurs peuvent êtres prises.

Défintion 1. *Soit $\mathcal{X}$=$\{X_1,\ldots,X_n\}$, D_i le domaine de la variable X_i et Doms=$\cup_{X_i\in\mathcal{X}} D_i$. Soit v_j∈Doms et l_j et u_j les bornes inférieures et supérieures de v_j. `Gcc`($\mathcal{X}$,[l_1,…,l_m],[u_1,…,u_m]) admet une solution ssi il existe une instanciation complète $\mathcal{A}$ telle que :*

$$\forall v_j \in Doms,\ l_j \leq |\ \{X_i \in \mathcal{X} \mid X_i = v_j\}\ | \leq u_j$$

La contrainte `Gcc` se modélise par un réseau pour lequel l'existence d'un flot maximal de valeur n permet de tester la cohérence ([15]). La recherche des composantes fortement connexes dans le graphe résiduel permet de filtrer toutes les valeurs non viables. Le test de cohérence se fait en $O(n\times m)$ et le filtrage en $O(n+m)$ (avec m égal au nombre d'arcs du réseau).

Enfin, toute contrainte `Gcc`(X,l,u) peut se décomposer en une conjonction de contraintes `Atleast` et `Atmost` portant sur les valeurs de $Doms$:
$\bigwedge_{v_j\in Doms}(\texttt{Atleast}(X,v_j,l_j) \wedge \texttt{Atmost}(X,v_j,u_j))$

3.2.2 La contrainte globale Σ-Gcc

Σ-`Gcc` est une version relaxée de `Gcc` selon la sémantique de violation basée décomposition notée μ_{dec} [10]. Σ-`Gcc` autorise le non respect des bornes inférieures et supérieures moyennant un coût de violation qui est fonction de l'écart aux bornes et du poids associé à celles-ci.

À chaque valeur v_j est associée un poids $\varphi_j^{atleast}$ (resp. φ_j^{atmost}) à la borne inférieure l_j (resp. supérieure u_j). On calcule le manque (resp. l'excès) grâce à la fonction $s(\mathcal{X},v_j)$ (resp. $e(\mathcal{X},v_j)$).

$$s(\mathcal{X},v_j) = max(0, l_j - |\{X_i \in \mathcal{X} \mid X_i = v_j\}|)$$
$$e(\mathcal{X},v_j) = max(0, |\{X_i \in \mathcal{X} \mid X_i = v_j\}| - u_j)$$

La violation de la contrainte Σ-`Gcc` pour la sémantique basée décomposition μ_{dec} est définie par :

$$\mu_{dec}(\mathcal{X}) = \sum_{v_j\in Doms} (s(\mathcal{X},v_j)\times\varphi_j^{atleast} + e(\mathcal{X},v_j)\times\varphi_j^{atmost})$$

Défintion 2 ([10])**.** *Soit z une variable objectif. Une contrainte Σ-`Gcc`($\mathcal{X},l,u,\varphi^{atleast},\varphi^{atmost},z$) admet une solution ssi il existe une instanciation complète $\mathcal{A}$ telle que $\mu_{dec}(\mathcal{A}) \leq max(D_z)$*

La modélisation d'une contrainte Σ-`Gcc` [10] s'effectue en ajoutant au réseau de `Gcc` des arcs de violation traduisant le manque ou l'excès pour chaque valeur. Ainsi sont ajoutés les arcs de violation suivants :

$$A_{shortage} = \{(s,v_j) \text{ avec } d=0, c=l_j, w=\varphi_j^{atleast} \mid v_j \in Doms\}$$
$$A_{excess} = \{(v_j,t) \text{ avec } d=0, c=\infty, w=\varphi_j^{atmost} \mid v_j \in Doms\}$$

L'existence d'un flot de valeur $max(n,\sum_{v_j\in Doms} l_j)$ de poids inférieur à $max(D_z)$ dans le nouveau réseau permet de caractériser la cohérence de Σ-`Gcc`. De la même manière, on peut caractériser la viabilité d'une valeur (X_i,v_j) par l'existence d'un flot passant par l'arc (X_i,v_j) de poids inférieur à $max(D_z)$. Le test de cohérence peut être réalisé en $O(max(n,\sum_{v_j\in Doms} l_j)\times n\cdot log(n))$ et le filtrage en $O(n\cdot d\times n\cdot log(n))$ [10].

Enfin, la contrainte `softGcc` [18] est un cas particulier de Σ-`Gcc` où tous les poids $\varphi_j^{atleast}$ et φ_j^{atmost} sont égaux à 1.

3.3 Relaxer la contrainte globale Regular

3.4 Regular

Soit M=$\{Q,\Sigma,\delta,q_0,F\}$ un automate fini déterministe avec Q l'ensemble des états, Σ un alphabet, δ l'ensemble des transitions définies sur $Q\times\Sigma\to Q$, q_0 l'état initial et F l'ensemble des états finaux de M. La contrainte `Regular`[13] associée à l'automate M impose qu'un mot constitué d'une séquence de n variables appartienne au langage reconnu par l'automate M.

Défintion 3. *Soit M un automate déterministe, $L(M)$ le langage régulier associé à M et $\mathcal{X}$=$X_1,\ldots,X_n$ une séquence de n variables. La contrainte `Regular`($\mathcal{X}$,M) admet un solution ssi :*

$$\exists (v_1,\ldots,v_n) \in D_1\times\cdots\times D_n \text{ t.q. } (v_1,\ldots,v_n) \in L(M)$$

La contrainte `Regular` est modélisée par un graphe dirigé de couches dont les sommets de chaque couche correspondent aux états de l'automate M et les arcs représentent les couples (variable/valeur) [13].

Le graphe en couches $\mathcal{G}$ est définit de la manière suivante :

$$\mathcal{G} = \{V, A\}$$
$$V = V_0 \cup \cdots \cup V_n \cup \{t\}$$
$$A = A_0 \cup \ldots A_n \cup A_t$$
$$\forall i \in \{0,\ldots,n\}, \text{ on a :}$$
$$V_i = \{q_l^i \mid q_l \in Q\}$$
$$A_i = \{(q_l^i, q_m^{i+1}, v) \mid v \in D_i, \delta(q_l,v) = q_m\}$$
$$A_t = \{(q_l^n, t) \mid q_l \in F\}$$

Dans ce graphe, tout chemin du sommet représentant l'état initial q_0^0 dans la première couche vers un sommet modélisant un état final q_i^n (avec q_i∈F) dans

la dernière couche correspond à une solution pour `Regular`. De la même manière, on peut caractériser la viabilité d'une valeur (X_i,v_j) par l'existence d'un tel chemin passant par l'arc (X_i,v_j). Le test de cohérence et le filtrage sont mis en œuvre par un parcours en largeur du graphe en $O(m)$ (avec m égal au nombre d'arcs du graphe).

3.5 costRegular

`costRegular` [2] est une version relaxée de `Regular` qui permet de modéliser le fait que certaines transitions de l'automate engendrent un coût si elles sont utilisées. Du point de vue de la modélisation, les coûts de ces transitions sont directement reportés sur les arcs associés dans le graphe de couches. Pour la sémantique de violation retenue μ_{reg}, le coût d'une instanciation complète $\mathcal{A}$ est la somme des coûts des arcs appartenant au chemin associé à cette instanciation

Défintion 4. *Soit M un automate déterministe et soit z une variable objectif. Une contrainte* `costRegular`*$(\mathcal{X},M,z)$ admet une solution ssi il existe une instanciation complète $\mathcal{A}$ telle que $\mu_{reg}(\mathcal{A}) \leq max(D_z)$.*

Le test de cohérence, associé à la sémantique de violation μ_{reg}, se ramène à la recherche d'un chemin de poids minimal dans le graphe associé. De même, on peut caractériser la viabilité d'une valeur (X_i,v_j) par l'existence d'un chemin de poids inférieur à $max(D_z)$ passant par l'arc (X_i,v_j). Le nouveau graphe de couches restant acyclique, on peut à nouveau utiliser un parcours en largeur du graphe. Ainsi le test de cohérence et le filtrage peuvent être effectués en $O(m)$.

4 Modélisation du problème

Dans cette section, nous montrons comment le problème décrit en Section 2 peut être modélisé de façon concise et élégante à l'aide des contraintes globales relaxées. De manière générale, on peut distinguer 2 sortes de règles :

- celles qui imposent des bornes sur le nombre d'infimières, d'équipes, ..., qui seront traduites par des contraintes `Gcc` ou `Σ-Gcc` ;
- celles qui imposent, pour chaque infirmière, des contraintes sur l'enchaînement des équipes qui seront traduites par des contraintes `Regular` ou `costRegular`.

4.1 Variables et domaines

Soit l'ensemble des jours J={1,2,...,21} et l'ensemble des infirmières I={1,2,...,8}. À chaque infirmière $i \in I$ et chaque jour $j \in J$ est associée une variable X_j^i. Tous les domaines initiaux sont identiques $D_i^j = \{M, E, N, R\}$, et les valeurs de *Doms* seront considérées dans cet ordre pour les contraintes `Gcc`[3]. Il en sera de même pour les bornes et les coûts associés.

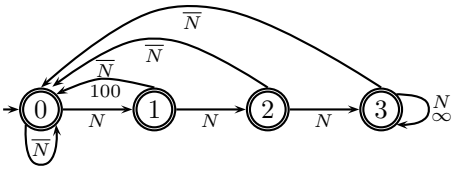

FIG. 1 – Automate A_1

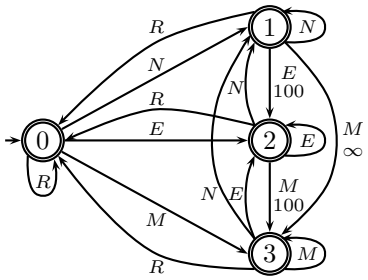

FIG. 2 – Automate A_2

4.2 Contraintes d'équipes

Les contraintes issues de la règle (I1) définissent les capacités de chaque équipe et seront modélisées grâce à la contrainte `Gcc`.

$$\boxed{\forall j \in J,\ \texttt{Gcc}([X_j^1,\ldots,X_j^8],[2,2,1,0],[2,2,1,8])}$$

4.3 Contraintes d'infirmières

i) Les règles suivantes peuvent être exprimées sous forme de `costRegular` en définissant les automates[4] suivants :

- l'automate A_1 associé aux règles (I5) et (P4) restreignant les enchaînements d'équipes de nuits (contrainte α) ;
- l'automate A_2 représentant les règles (I6) et (P5) contraignant l'écart minimal entre deux journées travaillées (contrainte β) ;
- l'automate A_3 décrivant les régles (P1) et (P3) gérant la longueur d'une séquence de travail (contrainte γ).

On obtient alors les contraintes suivantes :

$$\boxed{\begin{array}{ll} \forall i \in I,\ \texttt{costRegular}([X_1^i,\ldots,X_{21}^i],A_1,z_i^{A_1}), & (\alpha) \\ \forall i \in I,\ \texttt{costRegular}([X_1^i,\ldots,X_{21}^i],A_2,z_i^{A_2}), & (\beta) \\ \forall i \in I,\ \texttt{costRegular}([X_1^i,\ldots,X_{21}^i],A_3,z_i^{A_3}), & (\gamma) \end{array}}$$

[3] i.e. l=[l_M,l_E,l_N,l_R]

[4] La notation $\overline{N}$ signifie $\Sigma\backslash\{N\}$

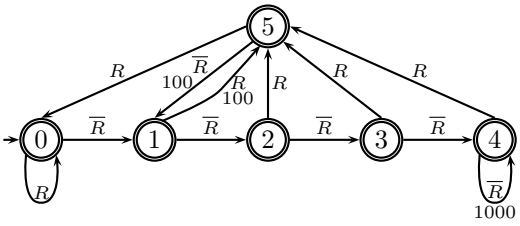

FIG. 3 – Automate A_3

ii) Les règles (I2) et (I3), qui sont des contraintes d'intégrité, peuvent être exprimées sous forme de `Gcc`. Toutefois, en associant des coûts infinis à ces contraintes, les règles (I2), (I3) et (P2) se modélisent grâce à un Σ-`Gcc` :

$$\forall i \in I,\ \Sigma\text{-}\texttt{Gcc}([X_1^i,\ldots,X_{21}^i],[4,4,0,7],\ [8,8,4,21],[10,10,0,\infty],[10,10,\infty,0],z_i)$$

La règle (I2) ne définissant que la borne supérieure pour la valeur N (u_N=4), nous imposons que la borne inférieure associée à N soit nulle (l_N=0) ainsi que son coût de violation ($\varphi_N^{atleast}$=0). De même, pour la borne supérieure de R, issue de la règle (I3), nous imposons u_R=21 et φ_R^{atmost}=0.

iii) La règle (I4) peut être exprimée en utilisant une contrainte `Atleast` :

$$\forall i \in I,\ \texttt{Atleast}([X_7^i,X_{14}^i,X_{21}^i],1,R)$$

4.4 Fonction objectif

Soit $\mathcal{C}$ l'ensemble de toutes les contraintes globales précédemment décrites : à chaque contrainte relaxée c est associée une variable objectif z_c permettant de quantifier le coût de violation de celle-ci.

5 Intéraction entre contraintes globales

Malgré la puissance de filtrage de chaque contrainte globale, le manque de communication entre celles-ci réduit la qualité du filtrage final. En effet, chaque contrainte globale travaille à partir de sa représentation interne (graphe bipartite, réseau, ...) et n'exploite pas (ou que partiellement) les informations déduites par les autres contraintes globales. Dans les NRPs, un grand nombre de contraintes globales partagent des ensembles communs de variables. La plupart des contraintes d'infirmières portent sur l'intégralité du planning d'une infirmière.

Divers travaux ont déjà été menés pour exploiter l'interaction entre contraintes globales :

- `Cardinality Matrix Constraints` [16] combinant plusieurs `Gcc` sous forme d'une matrice ;
- `multi-costRegular` [8] fusionnant plusieurs `costRegular`.

Dans cette section, nous proposons une nouvelle contrainte globale `RegularCount` (et sa version pondérée) permettant de combiner une contrainte `Regular` avec plusieurs contraintes `Atleast`/`Atmost` portant sur une même valeur.

5.1 Exemple

Considérons les règles suivantes issues de l'exemple de la section 2 :

(I2) Une infirmière doit travailler au plus quatre nuits.

(I5) Une infirmière doit travailler au plus trois nuits consécutives.

(P4) Chaque nuit isolée est pénalisée d'un coût de 100.

Pour les besoins de l'exemple, nous considérons dans un premier temps la règle (P4) comme étant une contrainte d'intégrité :

(P*4) Une infirmière ne doit pas avoir de nuit isolée dans son planning.

Ainsi, la règle (I2) peut être modélisée grâce à une contrainte `Atmost` et les règles (I5) et (P*4) par un contrainte `Regular` :

$$\forall i \in I,\ \texttt{Atmost}([X_1^i,\ldots,X_7^i],N,4)$$
$$\forall i \in I,\ \texttt{Regular}([X_1^i,\ldots,X_7^i],A_1^\star)$$

L'automate $A_1^\star$ est obtenu en enlevant la transition de l'état 1 à l'état initial dans l'automate A_1.

Considérons à présent un planning de sept jours pour une infirmière i et les domaines réduits des variables associés : $D_{X_1^i}=D_{X_2^i}=D_{X_3^i}=\{N\}$, $D_{X_4^i}=\{R\}$ et $D_{X_5^i}=D_{X_6^i}=D_{X_7^i}=\{N,R\}$. En appliquant de manière successive les filtrages des contraintes `Atmost` et `Regular`, aucun retrait de valeur n'est effectué.

Mais la valeur (X_5^i,N) est non viable. Si X_5^i est instanciée à N alors X_6^i sera nécessairement instanciée à N (règle (P*4)), ce qui engendre une incohérence (la règle (I2) est insatisfaite). La valeur (X_5^i,N) aurait du être filtrée ce qui n'est pas le cas. Ce raisonnement peut être réitéré pour les valeurs (X_6^i,N) et (X_7^i,N).

Dans ce qui suit, nous montrons qu'en tenant compte des interactions entre les contraintes `Atmost` et `Regular`, il est possible de fournir un filtrage plus efficace.

5.2 La contrainte RegularCount

La contrainte `RegularCount` permet à une séquence de variables $\mathcal{X}$ de prendre des valeurs telles que le mot engendré appartient au langage reconnu par un automate, tout en respectant des bornes sur le nombre d'occurence d'une valeur v_j fixée.

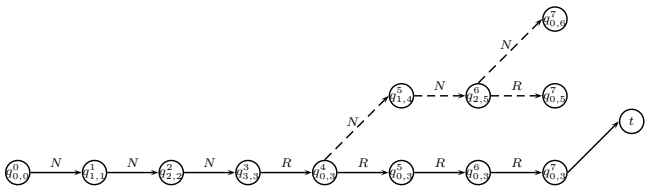

FIG. 4 – Réseau associé à l'exemple

Défintion 5. *Soit M un automate fini déterministe et $L(M)$ le langage reconnu par M. Soit $\mathcal{X}$=$X_1 \dots X_n$ une séquence de n variables, $v_j \in \Sigma$ et l_j (resp. u_j) une borne inférieure (resp. supérieure) sur le nombre de fois que la valeur v_j peut être prise. La contrainte* `RegularCount`*($\mathcal{X}$,M,v_j,[l_j,u_j]) admet une solution ssi :*

$$\exists (v_1, \dots, v_n) \in D_1 \times \dots \times D_n \text{ t.q. } (v_1, \dots, v_n) \in L(M) \\ \text{et } l_j \leq | \{X_i \in \mathcal{X} \mid X_i = v_j\} | \leq u_j$$

Ainsi, les règles (I2), (I5) et (P⋆4) peuvent être modélidées par la nouvelle contrainte `RegularCount` :

$\forall i \in I$, `RegularCount`($[X_1^i, \dots, X_7^i]$,$A_1^\star$,N,[0, 4])

5.2.1 Graphe associé à RegularCount

Soit v_j une valeur de Σ et l_j (resp. u_j) une borne inférieure (resp. supérieure) associée à la valeur v_j. L'idée principale est d'associer un compteur k à chaque sommet q_l^i de $\mathcal{G}$ pour comptabiliser le nombre de fois que la valeur v_j a été prise. Ainsi, à chaque sommet q_l^i sont associés k nouveaux sommets notés $q_{l,k}^i$, avec $k \in [0, \dots, n]$.

Les nouveaux sommets du graphe associé à la contrainte `RegularCount` sont définis par :

$$V = V_0 \cup \dots \cup V_n \cup \{t\}, \text{ avec :} \\ V_i = \{q_{l,k}^i \mid q_l \in Q, k \in [0..n]\}$$

Comme pour `Regular`, pour chaque transition entre deux couches du graphe est associé un arc reliant les sommets $q_{l,k}^i$ et $q_{m,k'}^{i+1}$ étiqueté par la valeur v_j. Toutefois, si à la valeur v_j est associé une paire de contraintes `Atmost`/`Atleast`, alors $k' = k + 1$ sinon $k' = k$. Finalement, les sommets de la dernière couche ayant une valeur de compteur respectant les bornes imposées par les contraintes `Atmost`/`Atleast` (et correspondant à des états finaux) seront connectés au puit :

$$A_i = \{(q_{l,k}^i, q_{m,k'}^{i+1}, v) \mid v \in D_i, \delta(q_l, v) = q_m\} \\ \text{si } (v = v_j) \text{ alors } k' = k + 1 \text{ sinon } k' = k \\ A_t = \{(q_{l,k}^n, t) \text{ si } q_l \in F \text{ et } l_j \leq k \leq u_j\}$$

La figure 4 représente le graphe associé à la contrainte `RegularCount`($[X_1^i, \dots, X_7^i]$,$A_1^\star$,N,[0, 4])) (exemple défini section 5.1).

5.2.2 Test de cohérence et filtrage

Le test de cohérence et le filtrage de `RegularCount` sont analogues à ceux de `Regular`. L'existence d'un chemin de $q_{0,0}^0$ à t permet de détecter la cohérence de la contrainte. L'existence d'un tel chemin utilisant l'arc (X_i,v) dans la couche correspondant à X_i permet de savoir si la valeur (X_i,v) est viable. Sur la figure 4, les arcs en pointillés indiquent les valeurs filtrées par la contrainte `RegularCount`. Ainsi, la valeur N est bien filtrée des domaines des variables X_5, X_6 et X_7 (exemple défini section 5.1).

5.3 La contrainte costRegularCount

Comme pour `CostRegular`, on souhaite modéliser le fait que les transitions impliquant une valeur particulière v_j engendrent un coût si elles sont utilisées. Les règles ci-dessous (issues de l'exemple de la section 2) permettent d'illuster une telle situation :

(I⋆2) Une infirmière doit travailler au plus quatre nuits, tout dépassement δ entraine une pénalité de $10 \times \delta$.

(I5) Une infirmière doit travailler au plus trois nuits consécutives.

(P4) Chaque nuit isolée est pénalisée d'un coût de 100.

Défintion 6. *Soit M un automate fini déterministe, $L(M)$ le langage associé, z une variable objectif. Soit $\mathcal{X}$ une séquence de n variables, $v_j \in \Sigma$, l_j (resp. u_j) la borne inférieure de poids $\varphi_j^{atleast}$ (resp. supérieure de poids φ_j^{atmost}) associé à v_j. La contrainte* `CostRegularCount`*($\mathcal{X}$,M,v_j,[l_j,u_j],[$\varphi_j^{atleast}$,φ_j^{atmost}],z) admet une solution ssi :*

$$\exists \mathcal{A} \in D_1 \times \dots \times D_n \text{ t.q } \mu_{reg}(\mathcal{A}) + s(\mathcal{A}, v_j) \times \varphi_j^{atleast} \\ + e(\mathcal{A}, v_j) \times \varphi_j^{atmost} \leq max(D_z)$$

Ainsi, les règles (I⋆2), (I5) et (P4) peuvent être modélisées par la contrainte `CostRegularCount` :

$\forall i \in I$:
`CostRegularCount`($[X_1^i, \dots, X_7^i]$,$A_1^\star$,N,[0, 4],[0, 10],$z_i^{A_1^\star}$))

Les sommets et les arcs du graphe correspondant à `CostRegularCount` sont construits de la même manière que pour `RegularCount`. Les arcs connectant les sommets de la dernière couche (correspondant à des états finaux) au sommet puits t sont remplacés par des arcs de violations permettant de modéliser le manque ou l'excès δ pour une valeur. Le coût de violation associé est $\delta \times \varphi^{atleast}$ (resp. $\delta \times \varphi^{atmost}$) pour la contrainte `Atleast` (resp. `Atmost`). Enfin, comme pour `CostRegular`, les coûts des transitions dans l'automate sont directement reportés sur les arcs associés dans

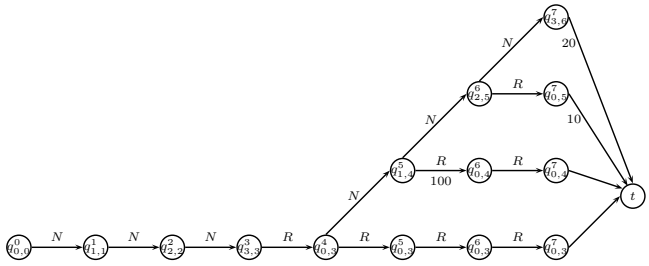

FIG. 5 – Réseau associé à l'exemple

le graphe de couches. La figure 5 donne le graphe correspondant à la contrainte `CostRegularCount`($[X_1^i,\ldots, X_7^i]$,$A_1^\star$,N,[0, 4],[0, 10],$z_i^{A_1^\star}$)).

Tester la cohérence revient à rechercher un chemin de poids inférieur à la valeur maximale de la variable objectif. Il en est de même pour le filtrage.

Complexité dans le pire cas : $O(n^2/2\times |\Sigma| \times |Q|)$

6 Variable Neighbourhood Search

La majorité des méthodes utilisées pour résoudre les NRPs sont soit des méthodes de RO ad'hoc, la plupart incluant des étapes de pré-taitement ou des méthodes de recherche locale combinant des techniques issues de la RO pour trouver une solution initiale (en résolvant à l'optimum un problème simplifié du problème de départ) pour la recherche locale.

Par nature, ces problèmes permettent de définir des structures de voisinages étendus (2-opt, swap de large portions des plannings d'infirmières,etc) en exploitant des informations spécifiques liées au problème.

La recherche à voisinage variable (VNS), [12], est une recherche à grands voisinages autorisant la taille de ceux-ci à varier afin de s'échapper des minima locaux. Pour cela, VNS utilise une *structure* de voisinage, c'est à dire un ensemble d'éléments $\{\mathcal{N}_1,\ldots,\mathcal{N}_{kmax}\}$ ordonnés par ordre croissant de la taille des voisinages engendrées.

Plusieurs versions de VNS ont été proposées. Variable Neighborhood Decomposition Search (VNDS), [3] permet d'effectuer une recherche uniquement sur un sous-problème déterminé en désaffectant un ensemble de k variables (k étant appelé la dimension du voisinage) et la reconstruction s'effectue alors uniquement sur le sous-espace de recherche contentant ces k variables. L'étape de reconstruction peut être effectuée par une recherche arborescente partielle.

6.1 VNS/LDS+CP

La méthode VNS/LDS+CP, [7], est une extension de l'algorithme VNDS effectuant la reconstruction des variables grâce à une méthode arborescente partielle (LDS) combinée avec des mécanismes de filtrage (CP). Toutefois, l'exploration de (très) grands voisinages pouvant rapidement devenir prohibitive en temps, nous avons retenu la recherche partielle LDS.

L'algorithme 1 décrit le pseudo-code de VNS/LDS+CP, avec $\mathcal{C}$ l'ensemble des contraintes, k_{init} (resp. k_{max}) le nombre minimal (resp. maximal) de variables à désaffecter et δ_{max} la valeur maximale de discrepancy lors de la phase de reconstruction avec LDS.

Algorithm 1: Algorithme VNS/LDS+CP.

```
1  function VNS/LDS+CP(X, C, k_init, k_max, δ_max)
2  begin
4      s ← genPremiereSolAvecLDS(X)
6      k ← k_init
8      while (k < k_max) ∧ (not timeout) do
10         X_unaffected ← H_vois(N_k, s)
12         A ← s\{(x_i = a) s.t. x_i ∈ X_unaffected}
14         s' ← NaryLDS(A, X_unaffected, δ_max, V(s), s)
16         if V(s') < V(s) then
18             s ← s'
20             k ← k_init
22         else k ← k + 1
24     return s
25 end
```

L'algorithme part d'une solution initiale s générée par la méthode LDS. Un sous-ensemble de k variables (avec k la dimension du voisinage) est sélectionné dans le voisinage N_k (i.e. l'ensemble des combinaisons de k variables parmi $\mathcal{X}$)(ligne 10). Une affectation partielle $\mathcal{A}$ est alors générée à partir de la solution courante s, en désaffectant les k variables sélectionnées ; les autres variables (i.e. non sélectionnées) gardent leur affectation dans s (ligne 12). $\mathcal{A}$ est alors reconstruite en combinant LDS et le filtrage. Si LDS trouve une solution voisine s' de meilleure qualité que la solution courante s (ligne 16), alors celle-ci devient la solution courante et k est réinitialisée à k_{init} (lignes 18-20). Sinon, k est incrémentée de 1 afin de s'échapper de ce minimum local(ligne 22). L'algorithme s'arrête dès que l'on a atteint la dimension maximale du voisinage à considérer k_{max} ou le timeout (ligne 8).

6.2 Gestion du voisinage

L'heuristique de choix de voisinage joue un rôle primordial dans la recherche, puisqu'elle détermine les sous-espaces à explorer afin de trouver des solutions de meilleure qualité. Notre heuristique de choix voisinage consiste à désaffecter complètement le planning d'une infirmière. Deux stratégies ont été mises en œuvre :

– l'heuristique `alea` basée sur un *tirage aléatoire* sur l'ensemble de tous les infirmières,

– l'heuristique `maxV` qui choisit l'infirmière dont le planning complet engendre le coût de violation le plus élevé.

6.3 LDS+CP

Notre heuristique de choix de variable sélectionne les variables par ordre croissant du ratio entre la taille du domaine et le degré. Les valeurs sont ordonnées selon l'ordre croissant des coûts engendrés par leurs affectations.

Le filtrage utilisé est celui des contraintes globales.

7 Expérimentations

Le solveur utilisé pour mener ces expérimentations a été développé en C++ et les tests ont été effectués sur PC P4-2.8Ghz. Les méthodes avec lesquelles nous nous comparons ayant été testées sur différents types de machines, nous exprimerons les temps de calcul obtenus en tenant compte du rapport de puissance entre la machine utilisée et la nôtre. Ainsi, pour une machine κ fois moins puissante que la nôtre, nous reporterons le temps obtenu divisé par κ

Pour VNS, k_{init} est le nombre de jours compris dans le planning d'une infirmière, et pour k_{max} 66% du nombre total de variables d'une instance. La discrepency est initialement fixée à 3 et la méthode est relancée avec une discrepancy augmentée après avoir exploré le voisinage de plus grande taille.

Les instances étudiées sont les suivantes :
– l'instance MILLAR [11] : 8 infirmères, 14 jours et 2 équipes possibles ;
– l'instance LLR [6] : 27 infirmières, 7 jours et 3 équipes possibles ;
– l'instance GPOST [4] : 8 infirmières, 28 jours et 2 équipes possibles ;
– l'instance IKEGAMI-3SHIFTS-DATA1 [5] : 25 infirmières, 30 jours et 3 équipes possibles.

N.B. : l'axe des ordonnées représente la valuation et celui des abscisses le temps en secondes.

7.1 Comparaison du filtrage de RegularCount

Nous avons modélisé chacune des instances LLR et GPOST de 2 façons différentes : à l'aide de `Atleast/Atmost+Regular` et à l'aide de `RegularCount`.

Comme le montrent les résultats présentés dans les figures 6 et 7, la contrainte `RegularCount` permet d'améliorer les performances de la recherche. Cette amélioration des performances est plus importante sur l'instance GPOST : cela est sûrement du au fait que les contraintes `RegularCount` remplacent dans GPOST un ensemble de contraintes `Regular/Atmost` portant sur 28 jours (et donc 28 variables) alors qu'elles ne portent que sur 7 jours dans l'instance LLR.

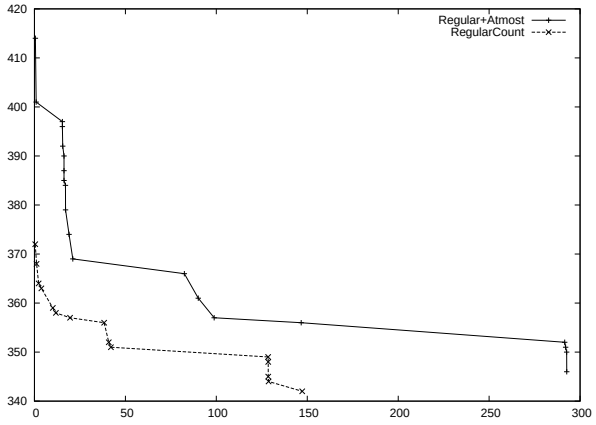

FIG. 6 – Comparaison de `Regular + Atmost/Atleast` et `RegularCount` sur l'instance LLR

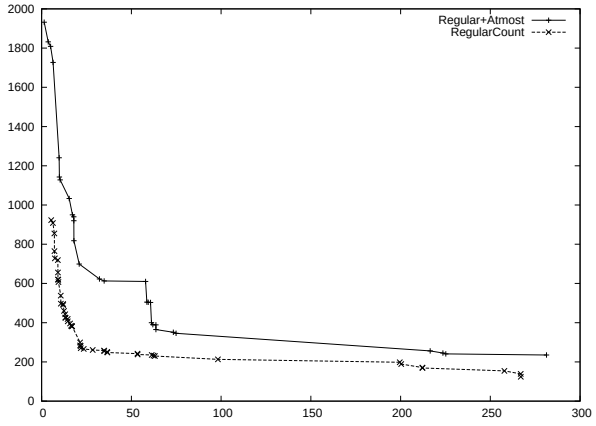

FIG. 7 – Comparaison de `Regular + Atmost` et `RegularCount` sur l'instance GPOST

7.2 Comparaison avec une approche CSP

Dans [6] les auteurs proposent une approche basée CSP, pour l'instance LLR, qui utilise des contraintes binaires et des propagateurs spécifiques pour les contraintes n-aires.

La recherche s'effectue en deux étapes. Premièrement, une solution initiale est calculée en ne tenant compte que des contraintes d'intégrité. Dans une seconde temps, toutes les contraintes sont prises en compte et une procédure de recherche tabou améliore la solution initiale.

Grâce à cette méthode une solution de qualité 366 est trouvée en 16 secondes.

Nous trouvons une borne à 314 en environ 1 minute. Il est à noter que la première solution obtenue a pour qualité 372 et qu'une instanciation complète de coût

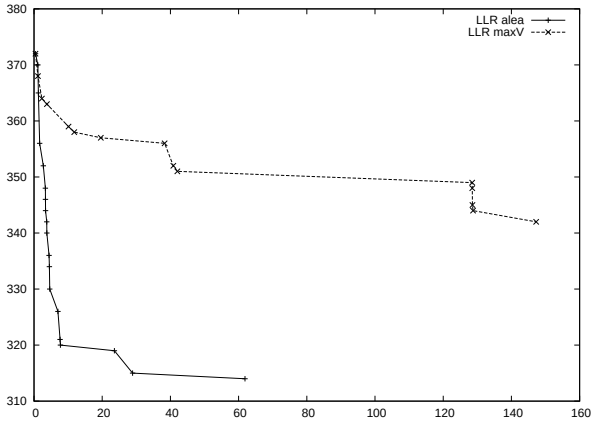

FIG. 8 – Instance LLR

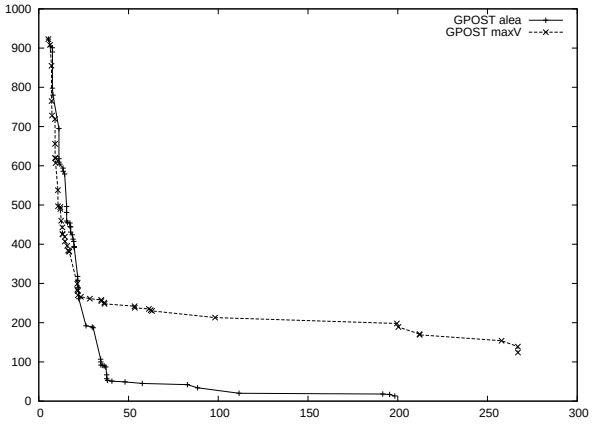

FIG. 9 – Instance GPOST

inférieur à 366 est obtenue en moins de 2 secondes avec les deux heuristiques de choix de voisinage : `alea` et `maxV`.

Ces résultats montrent que les contraintes globales relaxées permettent de trouver de bonnes premières solutions et d'améliorer rapidement la qualité de celles-ci grâce à la puissance de leur filtrage.

7.3 Comparaison avec des méthodes ad'hoc

7.3.1 L'instance MILLAR

Les premiers résultats obtenus pour MILLAR sont présentés dans [11]. La technique utilisée consiste à générer l'ensemble des motifs valides de longueur variant de 1 à 5 jours (une infirmières ne devant pas travailler plus de 5 jours consécutifs). À partir de ces motifs, un graphe dirigé est créé avec pour sommets les motifs et pour arcs les connexions entre les différents motifs. Ces arcs sont pondérés par le degré de violation provoqué par l'enchaînement de ces motifs.

Grâce à cette approche, une solution de coût 1690 est trouvée en environ 500 secondes sur IBM RISC6000/340[5].

Grâce à la résolution de sous-problème et d'une recherche tabou, une solution optimale (de coût 0) est trouvée en moins d'une seconde ([5]).

Notre solveur trouve l'optimum en 3 secondes.

NB : La borne supérieure utilisée dans [5] n'est pas indiquée, c'est pourquoi nous avons arbitrairement fixée notre borne initiale à 10000. Si celle-ci était fixée à une valeur plus faible, notre temps de calcul serait très fortement amélioré.

7.3.2 L'instance GPOST

Cette instance est résolue en 2 étapes (voir [4]). Dans un premier temps l'ensemble des plannings possibles pour chaque infirmière est calculé. Parmi ces plannings, seuls sont conservés ceux qui ont un degré de violation assez faible. Le planning général est alors calculé grâce à CPLEX. Le temps "d'assemblage" du planning général optimal est de 13 secondes (coût 5).

Pour cette instance nous avons borné le temps d'éxecution de notre solveur à 300 secondes. Nous obtenons les résultats présentés dans la figure 9.

Notre approche obtient en moins de 200 secondes une solution de coût 13.

Il est assez difficile de comparer nos temps de calcul avec ceux proposés dans [4] sans connaître le temps utilisé pour l'étape de prétraitement.

7.3.3 L'instance IKEGAMI-3SHIFTS-DATA1

Dans [5], avec une approche similaire à celle de MILLAR, les auteurs obtiennent une solution de coût 6 en 1851 minutes.

Pour des raisons pratiques, nous avons fixé le temps maximal de recherche à 1 heure.

Malgré la grande taille de cette instance (750 variables), les résultats présentés sont très encourageants (figure 10). En effet, l'heuristique de voisinage `maxV` nous fournit en 11 minutes et 11 secondes une solution de coût 63 . Cette solution est de bonne qualité car seules des contraintes de poids 1 sont insatisfaites.

8 Conclusions

Dans cet article, nous avons montré comment certains NRPs peuvent être modélisés à l'aide de contraintes globales relaxées et résolus dans des temps

[5]Le temps ici n'est pas normalisé

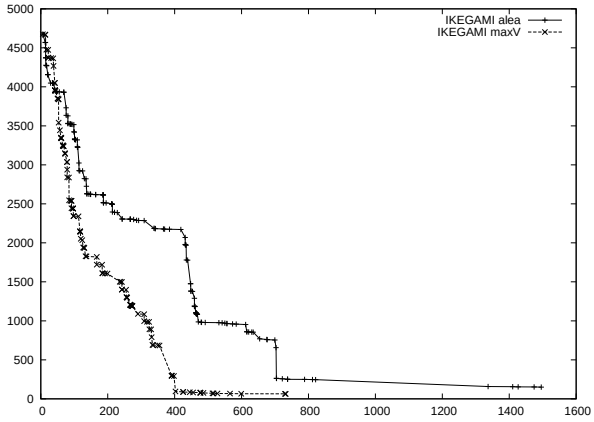

FIG. 10 – Instance IKEGAMI

comparables aux méthodes ad'hoc utilisées. Ces résultats sont à conforter par des expérimentations sur des instances de très grande taille (les instances : CHILD-A2 (41 infirmières × 42 jours × 5 équipes), ERRVH-A (51 infirmières × 42 jours × 8 équipes), ...). Les performances de notre méthode de recherche peuvent être améliorées i) en définissant des heuristiques de sélection de voisinage plus adaptées aux NRP (par exemple, ne pas désinstancier toutes les variables associées à une même infirmière), ii) en étendant la notion de soft arc-conhérence introduite par [1] pour les contraintes binaires afin d'améliorer la communication entre les contraintes globales.

Références

[1] Martin C. Cooper, Simon de Givry, Martí Sánchez, Thomas Schiex, and Matthias Zytnicki. Virtual arc consistency for weighted csp. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 253–258. AAAI Press, 2008.

[2] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4) :315–333, 2006.

[3] P. Hansen, N. Mladenovic, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4) :335–350, 2001.

[4] http ://www.cs.nott.ac.uk/ tec/NRP/.

[5] Atsuko Ikegami and Akira Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3) :517–541, 2003.

[6] Haibing Li, Andrew Lim, and Brian Rodrigues. A hybrid ai approach for nurse rostering problem. In *SAC*, pages 730–735. ACM, 2003.

[7] Samir Loudni and Patrice Boizumault. Combining vns with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191(3) :705–735, 2008.

[8] Julien Menana, Sophie Demassey, and Narendra Jussien. Relaxation lagrangienne pour le filtrage d'une contrainte-automate à coûts multiples. In Ammar Oulamara, editor, *ROADEF*, pages 43–44, 2009.

[9] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Σ-alldifferent : Softening alldifferent in weighted csps. In *ICTAI (1)*, pages 223–230. IEEE Computer Society, 2007.

[10] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Softening gcc and regular with preferences. In *SAC*, volume 3, pages 1392–1396, 2009.

[11] Harvey H Millar and Mona Kiragu. Cyclic and non-cyclic sheduling of 12h shift nurses by network programming. *European Journal of Operational Research*, 104 :582–592, 1996.

[12] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers And Operations Research*, 24 :1097–1100, 1997.

[13] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

[14] T. Petit, J-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *CP'01*, volume 2239 of *LNCS*, pages 451–463, 2001.

[15] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *AAAI/IAAI, Vol. 1*, pages 209–215, 1996.

[16] Jean-Charles Régin and Carla P. Gomes. The cardinality matrix constraint. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 572–587. Springer, 2004.

[17] Thomas Schiex. Soft constraint processing. *First International Summer School on Constraint Programming*, 2005.

[18] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming : Flow-based soft global constraints. *J. Heuristics*, 12(4-5) :347–373, 2006.