Intégration des méthodes de réécriture et de recherche opérationnelle pour la modélisation et la résolution de contraintes : application à la planification de personnel médical

Nadia Brauner, Rachid Echahed, Gerd Finke, Frederic Prost, Wendelin Serwe

Laboratoire Leibniz-IMAG
46, av. Félix Viallet
38031 GRENOBLE Cedex
France
{Nadia.Brauner, Rachid.Echahed, Gerd.Finke, Frederic.Prost, Wendelin.Serwe}@imag.fr

Résumé. Nous présentons un travail où deux techniques différentes, la Recherche Opérationnelle et les techniques de réécritures sont couplées afin de résoudre des problèmes de gestions de ressources. L'idée générale est que si la R.O. fourni des algorithmes efficaces, ces derniers ne sont valides que sur des structures mathématique précises or il existe des contraintes symboliques difficilement représentables de cette manière. Nous proposons donc d'utiliser les techniques de réécritures, qui sont générales mais gourmandes en temps, pour traiter ces dernières et de se servir de la R.O. pour traiter la partie combinatoire du problème. Nous illustrons notre méthode en l'appliquant à la gestion des ressources humaines et matérielles d'un service de radiologie d'un hôpital parisien.

Mots clés: Emplois du temps, gestion des ressources humaines et matérielles, recherche opérationnelle, réécriture

1. Recherche opérationnelle et réécriture

L'objectif de ce travail est de modéliser et de résoudre simplement et rapidement des problèmes génériques ayant de nombreux types de contraintes souvent complexes. L'approche choisie est illustrée par l'application de la méthode à un cas réel : la planification de personnel médical dans un service de radiologie d'un hôpital parisien.

L'idée principale est d'intégrer les techniques de description de contraintes (langages de haut niveau, réécriture (Dershowitz et Jouannaud, 1990)) et de résolution de contraintes (méthodes de la recherche opérationnelle). En effet, d'un côté, l'étude des algorithmes de résolution de contraintes est l'apanage de la recherche opérationnelle (R.O.). De nombreuses méthodes ont été proposées sur des cas spécifiques pour traiter de manière efficace des contraintes sur des milliers de variables. Néanmoins, ces méthodes sont en général statiques et ne tiennent pas compte aisément des évolutions dynamiques des systèmes de contraintes. Par ailleurs, les langages de programmation de très haut niveau permettent des descriptions aisées et rapides de systèmes de contraintes statiques et dynamiques. Cependant, les méthodes utilisées en programmation pour la résolution de contraintes symboliques ou numériques ne sont pas toujours aussi efficaces que celles proposées par la recherche opérationnelle.

2. Étude de cas : planification de personnel médical

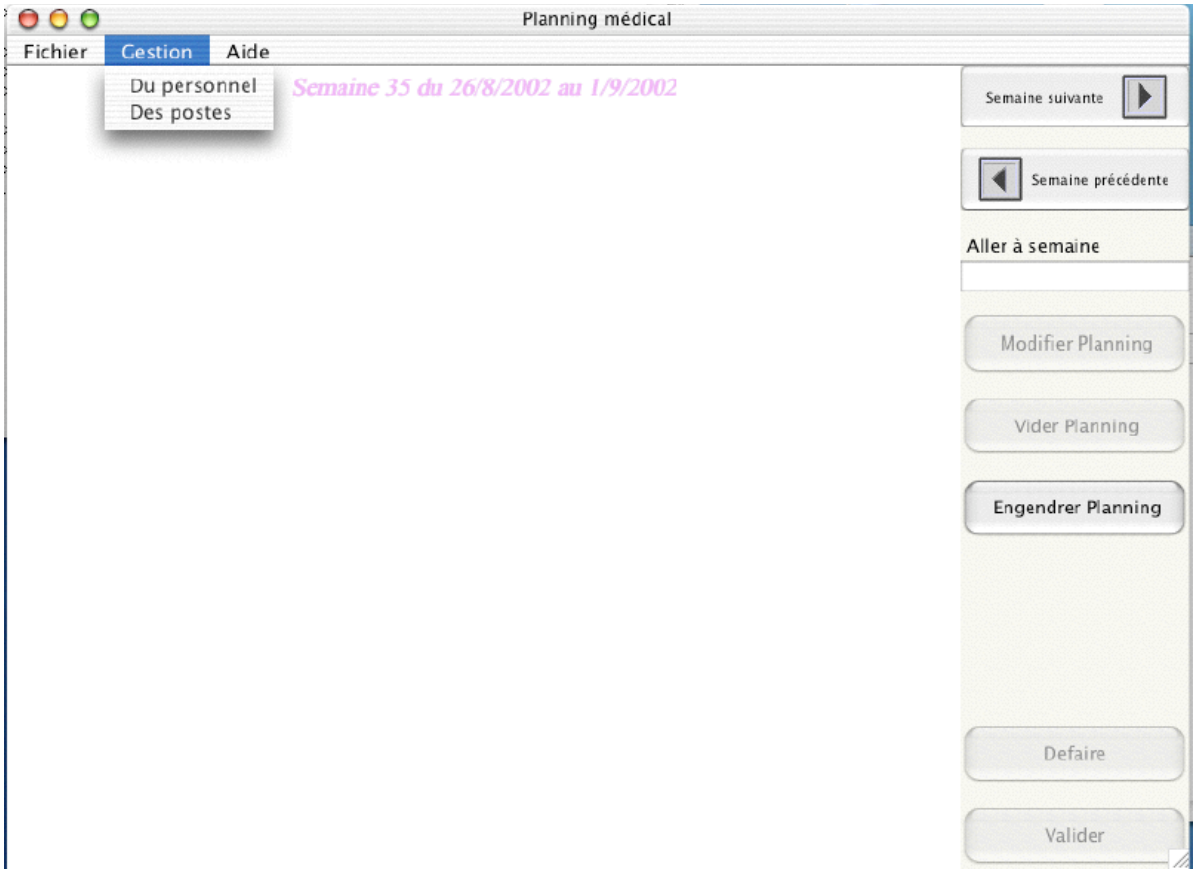
Pour illustrer notre méthode, nous considérons le problème concret de génération hebdomadaire d'emploi du temps de médecins dans un service de radiologie. La personne chargée de la planification dispose d'un certain nombre de postes et d'une équipe de médecins avec des compétences diverses à affecter à ces postes par demi-journées. Elle doit, de plus, prendre en compte de nombreuses contraintes de différents

types. Par exemple, elle dispose d'un planning d'ouverture des postes et de disponibilité des médecins. Sur une journée, sur certains postes, le médecin affecté doit être le même le matin et l'après-midi (urgences, par exemple) alors que sur d'autres postes, le médecin du matin doit être différent de celui de l'après-midi (écho1, par exemple). Chaque médecin exprime des préférences et est plus ou moins compétent pour travailler sur les différents postes. Ceci implique par exemple que certains médecins ne peuvent pas être seuls sur certains postes tant qu'ils n'ont pas été formés. De plus, un médecin ne peut pas être affecté plus de trois fois par semaine sur le même poste.

Ces exemples montrent que le problème possède des contraintes de types très variés. De plus, les niveaux de contraintes sont différents : certaines contraintes peuvent être violées alors que d'autres sont très fortes. La méthode doit également permettre une gestion dynamique des contraintes : les contraintes et les types de contraintes sont susceptibles de changer d'une semaine à l'autre.

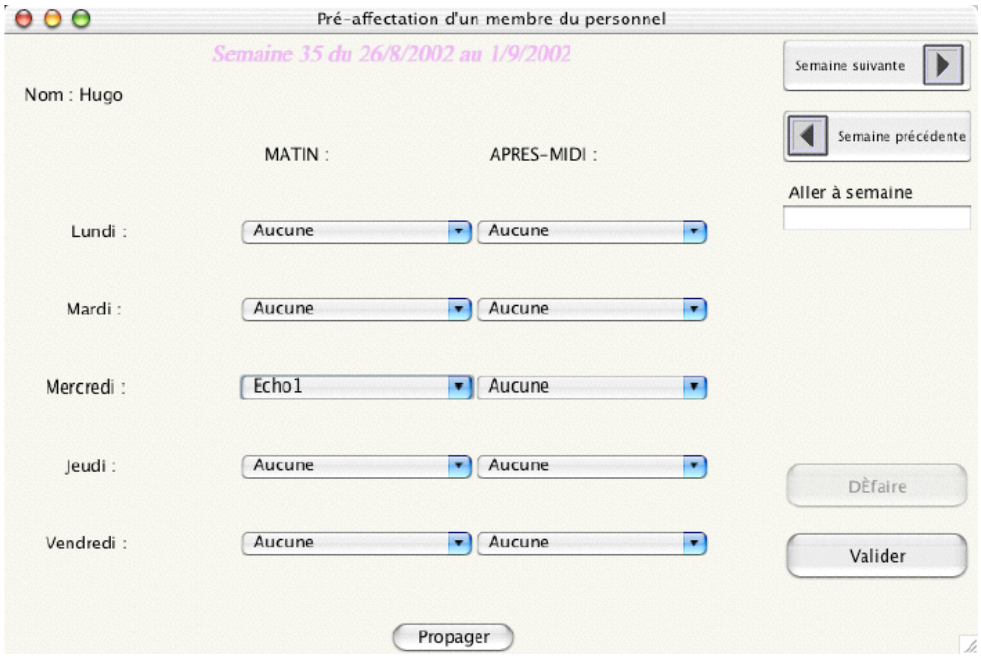
3. Description du logiciel

Afin de valider la méthode par la personne chargée de l'affectation du personnel, une interface graphique conviviale a été programmée en Java. Lors du lancement du logiciel, la fenêtre suivante apparaît. Les différents boutons permettent de se déplacer dans le temps et de gérer les plannings. Les menus permettent de gérer les médecins (ajout et retrait, noms, présences, pré-affectations, compétences…) et les postes (noms, ouvertures).



Pour chaque médecin, il est possible de définir un nom, une fonction et une couleur. Les médecins ont également des compétences, c'est-à-dire, qu'ils peuvent travailler sur certains postes et non sur d'autres. Une fois ces éléments précisés, la fenêtre suivante permet d'indiquer les disponibilités et les pré-affectations des médecins (par exemple, pour des vacations fixes). Ces pré-affectations peuvent être

propagées sur un certain nombre de semaines par exemple si un médecin doit être sur le poste «écho 1» tous les mercredis matins.



Pour les postes, il est possible de définir des ouvertures et de les propager sur plusieurs semaines consécutives (par exemple si un poste est systématiquement «prêté» à un autre hôpital tous les vendredis après-midi). La fenêtre suivantes indique par exemple que pendant la semaine 35, le poste radio est ouvert les mardi, jeudi et vendredi matins et lundi et jeudi après-midi.

Lorsque tous ces paramètres ont été définis, cliquer sur la touche «engendrer planning» de la fenêtre d'accueil envoie les informations au moteur qui calcule un planning. Ce planning est ensuite affiché comme indiqué dans la figure suivante:

4. Méthode de résolution

Les techniques de la R.O. sont développées en vue de résoudre efficacement des problèmes portant sur des structures mathématiques précises, par exemple l'algèbre linéaire, les graphes etc. Ces structures peuvent être utilisées moyennant un codage pour modéliser des problèmes réels. Il s'agit alors de transformer et d'adapter les données et les contraintes pour pouvoir utiliser les algorithmes de la R.O. Ainsi

dans notre étude de cas, le problème de planification peut, de manière globale, se voir comme un problème d'affectation médecin-poste-jour. De plus, certaines contraintes spécifiques sont d'un genre symbolique et sont difficiles et/ou non naturelles à coder de manière numérique. Par exemple, le fait qu'un médecin ne puisse être affecté plus de x fois à tel poste sur une période de temps donnée etc. Pour ce genre de contraintes, les systèmes de réécriture permettent une description aisée. De plus, ces descriptions en elles-mêmes forment un programme permettant de trouver toutes les solutions respectant ces contraintes.

L'idée de notre expérience est donc d'utiliser les systèmes de réécriture comme "enrobage" pour les algorithmes de R.O. Les techniques de R.O. seront concentrées sur des parties du problème purement combinatoires alors que le système de réécriture permettra de traiter les autres contraintes qui ne s'expriment pas facilement sous forme utilisable directement en R.O.

Le système de réécriture.

Informellement, un système de réécriture de termes (Dershowitz, Jouannaud, 1990) se présente comme une collection de règles de la forme : `lhs => rhs if c`, signifiant que la partie gauche `lhs` se réécrit en `rhs` si la condition `c` est vérifiée. S'il n'y a pas de condition, on écrit simplement `lhs => rhs` comme dans le système de réécriture suivant :

```
(0) 0 + x => x
(1) Succ(x) + y => Succ(x+y)
```

où `Succ` dénote la fonction successeur qui ajoute un à un entier (par exemple `Succ(2)=3`). La règle `(0)` énonce juste qu'ajouter un entier à 0 a pour résultat cet entier. La règle `(1)` dit qu'ajouter un entier plus un à un autre revient à ajouter les deux entiers plus un. Exécuter un programme dans un système de réécriture revient à transformer un terme en utilisant les règles. En reprenant notre exemple :

```
Succ(Succ(0))+ Succ(0) => Succ(Succ(0)+Succ(0))
```

en utilisant la règle `(1)`, avec `x=Succ(0)` et `y=Succ(0)`. On peut réappliquer la règle (1) sur le terme résultat avec `x=0` et `y=Succ(0)`, ce qui nous donne :

```
Succ(Succ(0)+Succ(0)) => Succ(Succ(0+Succ(0)))
```

et finalement on voit qu'on peut appliquer la règle (0) pour conclure

```
Succ(Succ(0+Succ(0))) => Succ(Succ(Succ(0)))
```

autrement dit que (1+1+0)+(1) se calcule en (1+1+1) soit en 3.

Une autre manière d'utiliser les systèmes de réécriture consiste à résoudre des contraintes symboliques contenant des variables en utilisant un algorithme de résolution basé sur la *surréduction* (Hullot, 1980). Par exemple :

```
Succ(Succ(0)) + y = Succ(Succ(Succ(0)))
```

Ici la méthode de résolution symbolique va chercher la valeur de l'inconnue `y` pour laquelle cette équation sera valide vis-à-vis des règles de réécritures qui définissent le système.

Comme on le voit ce formalisme de réécriture est très général et permet naturellement l'expression de contraintes. Dans notre étude de cas on se sert d'un tel système pour décrire les contraintes du problème. Par exemple :

1) Pré-affectation des médecins, qui indique quels médecins sont présents sur une période donnée. Par exemple la règle

```
present (Medecin Gerd) Monday PM => true
```

indique que le médecin de nom `Gerd` est présent le mardi après-midi alors que la règle

```
present (Interne Rachid) Friday AM => false
```

indique que l'interne `Rachid` n'est pas présent le vendredi matin.

2) Horaires des postes, qui indiquent quand sont ouverts ou fermés les différents postes. Par exemple

```
open Echo1 Friday AM => true
```

indique que le poste d'écho1 est ouvert le vendredi matin.

3) Vérification qu'une affectation est correcte relativement aux contraintes particulières (pas le même médecin affecté le matin et l'après-midi sur le même poste etc.). Par exemple

```
unqualified (Medecin Frederic) Echo2 => true
```

indique que le médecin Frédéric n'est pas qualifié pour le poste d'écho2.

On pourrait aussi facilement représenter le problème de l'affectation proprement dite de cette manière. Cependant, la recherche de solutions dans un système de réécriture si elle est très générale n'est pas efficace. De plus, face à l'explosion combinatoire de ce problème cette recherche ne pourrait rendre de résultat en temps raisonnable.

Pour la résolution du problème de l'affectation de personnel, nous avons procédé de la façon suivante. La semaine est découpée en une liste de blocs (un bloc correspondant à une demi-journée), dans chacun des blocs on utilisera une méthode de la R.O. pour réaliser l'affectation, et on utilisera les règles de réécritures pour gérer les contraintes inter-blocs ainsi que les données d'initialisation (pré-affectation des médecins, emplois du temps des postes etc.).

Les règles qui sont appelées pour engendrer un emplois du temps ont la forme suivante :

```
timetable schedule => true if (schedule == init_schedule) and
                              (is_timetable schedule == true)
```

Ici le résultat recherché est la variable `schedule` (équivalent du `y` dans la contrainte `Succ(Succ(0)) + y = Succ(Succ(Succ(0)))`. On a rajouté les contraintes suivantes :

1) le résultat (`schedule`) doit vérifier les pré-affectations : `(schedule==init_schedule)`, et
2) le résultat doit représenter un emploi du temps valide : `(is_timetable schedule)`

En fait la condition (2) fait appel à une fonction qui va effectivement calculer l'emplois du temps. C'est elle qui, pour chaque bloc, fera appel a un module de R.O. (il s'agit de la méthode hongroise). Pour ce bloc, le module rendra une affectation localement correcte :

```
is_timetable([Block day period schedule|block_list]) => true if
     (is_timetable block_list == true) and
     (hungarian_linear_assignment (Block day period schedule))
```

Ainsi pour qu'une liste de blocs représente un emplois du temps correct il faut que pour chaque bloc (le terme `(is_timetable block_list)` représente le traitement du reste de la liste) on puisse faire une affectation en utilisant la méthode hongroise représentée dans la règle ci-dessus par la contrainte `(hungarian_linear_assignment (Block day period schedule))`.

Module de la RO

Ce module qui utilise la méthode hongroise d'affectation linéaire (Burkard et Derigs, 1980) considère un bloc, c'est-à-dire l'affectation des médecins aux postes pour une demi-journée. Plus formellement, d'une part, nous avons m postes $P_j, j = 1,2...m$, d'autre part, les médecins $M_i, i = 1,2...n \quad (n \geq m)$. En ajoutant $n-m$ postes fictifs $P_{m+1}, P_{m+2}...P_n$, une affectation médecin-poste x_{ij} correspond à une solution des équations suivantes

$$\sum_{i=1}^{n} x_{ij} = 1 \quad , \quad j = 1,2...n$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad , \quad i = 1,2...n$$

$$x_{ij} = 0 \; ou \; 1$$

Les $n-m$ médecins affectés à un poste fictif sont libres, cette demi-journée pour effectuer leur recherche ou pour être en renfort sur un poste déjà affecté. Un fonction coût $\sum c_{ij} x_{ij}$ dirige la recherche d'un emploi du temps admissible.

La matrice des coûts (c_{ij}) est calculée par le système de réécriture. Les indisponibilités des médecins et les fermetures des postes correspondent à des coûts infinis. Les autres coûts dépendent des blocs déjà traités. Une granularité plus fine des coûts est aussi envisageable si l'on souhaite intégrer des compétences multi-niveaux des médecins.

5. Conclusion

Nous avons proposé une intégration de l'algorithme d'affectation linéaire et des systèmes de réécriture. Le formalisme résultant permet de décrire aisément les problèmes liés à la planification du personnel. L'implantation que nous avons faites (interface en Java, moteur en Sabir (Echahed et Serwe, 2000)) donne un emploi du temps sur un PC standard en environ 3 minutes pour ce type de problèmes.

Nous envisageons plusieurs extensions de nos travaux. D'une part, l'algorithme hongrois fournit une seule solution optimale. Mais dans certains cas, il est nécessaire de se contenter d'une solution localement sous-optimale afin de pouvoir trouver une solution globale. D'autre part, il semble intéressant de considérer d'autres algorithmes de la R.O. à intégrer aux systèmes de réécriture afin de fournir une bibliothèque d'algorithmes qui permet le traitement d'une large classe de problèmes.

6. Références

R.E. Burkard et U. Derigs (1980). Assignment and matching problems: Solution methods with Fortran programs, volume 184 of Lecture Notes in Economics and Mathematical Systems. Springer, Berlin.

N. Dershowitz et J.-P. Jouannaud (1990). Rewrite Systems. Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. pages 243-320, chapitre 6. Elsevier.

R. Echahed et W. Serwe (2000). Combining Mobile Processes and Declarative Programming. Dans J. Lloyd et al. (ed.), actes de CL2000, pages 300-314, LNAI 1861, Springer.

J.-M. Hullot (1980). Canonical forms and unification. In Proc of 5th Conference on Automated Deduction LNCS 87, Springer Verlag, pp 318-334.