Programmation Par Contraintes - TP Choco
Projet 1: Nurse Scheduling Problem

Sophie Demassey

19 novembre 2011

A Présentation du projet

A.1 Organisation

- séance 1 (TD/TP – 2h30) : découverte du problème, conception du modèle de base
- pour la séance suivante : implémentation Choco du modèle de base
- séance 2 (TD/TP – 2h30) : amélioration du modèle, étude et implémentation
- pour la séance suivante : terminer l'implémentation
- séance 3 (TD/TP – 2h30) : heuristiques de branchement, expérimentations
- pour la séance suivante : tests benchmarks, préparation du rendu
- 6ème semaine : livraison du prototype

A.2 Objectifs de formation

Programmation par contraintes
- conception d'un modèle de contraintes à partir de spécifications concrètes
- amélioration de modèles de contraintes : degré de consistance, contraintes redondantes, conjonction de contraintes

Implémentation / solveur Choco
- manipulation des éléments de base de Choco, lecture et recherche de documentation
- définition de modèles Choco avec contraintes globales
- résolution avec heuristiques de branchement

Ingénierie
- analyse des spécifications
- développement logiciel (programmation objet, qualité logicielle, documentation)
- expérimentation et analyse des résultats

B Présentation du problème

Le Nurse Scheduling Problem (NSP) est un problème de planification de personnel typique des services hospitaliers. Dans un tel service, les gardes des infirmières sont organisées sur la base de *quarts* de travail, tels que le quart de matin (5h-14h), de journée (8h-18h), de nuit (18h-8h). Calculé à partir d'une estimation de la charge de travail attendue, le nombre requis de gardes sur chaque quart de chaque jour est connu. Établir l'emploi du temps du personnel du service sur une période donnée, par exemple 1 mois, consiste à décrire les jours de repos et les quarts de travail opérés par chaque infirmière et pour chaque jour de la période. L'emploi du temps est valide, d'une part, s'il couvre tous les requis en nombres de gardes, et d'autre part, s'il respecte les règles de travail individuellement pour chaque infirmière. Ces différentes règles proviennent du code de la réglementation du travail, du contrat de travail lié à la fonction de l'infirmière, ou encore des choix de garde ou de congé de l'infirmière.

Un logiciel permettant d'automatiser la planification de personnel doit tenir compte *de la variabilité des règles de travail* – elles changent à chaque nouvelle réglementation, elles sont propres à chaque pays, hôpital, service. Pour être générique, un tel logiciel doit offrir une grande flexibilité dans la modélisation des contraintes. Pour être performant, il doit également tenir compte *de la forte interaction entre les multiples règles* présentes au sein d'une même instance, et qui rend cette dernière difficile à résoudre.

Dans cet exercice, vous allez concevoir le prototype d'un logiciel de résolution de NSP, en vous basant sur l'étude d'une instance particulière. Votre prototype devra être orienté à la fois performance (rapidité d'exécution) et généricité (applicable à d'autres instances).

B.1 (NSP_1) : Un premier jeu de données

On considère un service de garde hospitalière composé de 8 infirmières et organisé en seulement 2 types de quarts de travail : JOURNÉE (J) et NUIT (N). La planification est effectuée sur une période d'1 mois comprenant 28 jours, tous ouvrables : du lundi 1er février et dimanche 28 février. Sur cette période, le nombre de gardes requis est constant : 3 gardes de JOURNÉE et 1 garde de NUIT par jour. Dans ce service, il existe deux types de contrats : 4 infirmières (F1, F2, F3, F4) sont en contrat à temps plein, les 4 autres (P1, P2, P3, P4) sont en contrat à temps partiel. Un contrat à temps plein spécifie que l'infirmière travaille exactement 18 jours par mois, dont au plus 4 quarts de NUIT, et entre 4 et 5 jours par semaine (du lundi au dimanche). Un contrat à temps partiel spécifie que l'infirmière travaille au plus 10 jours par mois, dont au plus 4 quarts de NUIT, et entre 2 et 3 jours par semaine. La réglementation générale du service impose, pour tout son personnel, les règles suivantes :

1. il est interdit de travailler plus de 5 jours consécutifs ;
2. un week-end (samedi et dimanche) est soit travaillé les 2 jours, soit chômé les 2 jours ;
3. il est interdit de travailler de nuit avant un week-end chômé ;
4. exactement 2 jours de repos sont requis après un quart de NUIT ou une série de quarts de NUIT ;
5. il est interdit de travailler plus de 2 week-ends consécutifs.

Enfin, compte tenu de la planification de janvier, la planification des deux premiers jours du mois de février est imposée : les infirmières F2, P2, P3, P4 ne travaillent pas les 1er et 2 février ; les infirmières F1, F3 et P1 assurent les quarts de JOURNÉE les 1er et 2 février.

B.2 Terminologie

activité : types de quart ou de congé. En l'occurrence, (NSP_1) comporte 2 activités travaillées D (JOURNÉE/DAY) et N (NUIT/NIGHT) et une activité non-travaillée, notée R (REPOS/REST).
règle de couverture : limite sur le nombre de gardes pour un quart et un jour donnés
règle de travail : autre règle portant sur l'emploi du temps d'un employé, dont :
règle d'assignation : requis de garde/congé pour un jour donné
règle de cardinalité : limite sur le nombre de gardes/congés dans une plage de temps donnée
règle de séquencement : interdiction de séquence de gardes/congés sur des jours consécutifs

C Séance 1 : Modèle mathématique et modèle Choco.

C.1 Modélisation mathématique du problème de base.

Dans un premier temps, nous considérons la variante (NSP_0) simplifiée du problème (NSP_1), dans laquelle aucune règle de travail n'est présente : seules les règles de couverture sont considérées.

Question 1 *Proposer un modèle mathématique pour le problème* (NSP_0)*, exprimé au moyen d'opérateurs mathématiques courants (arithmétiques, ensemblistes, etc.). Quelle est la taille du modèle, i.e. : nombre de variables et de contraintes et arité des contraintes ?*

Question 2 *Proposez une représentation graphique de* (NSP_0) *et indiquez la classe de complexité de ce problème ? Décrivez un algorithme de résolution et donnez sa complexité.*

C.2 Modélisation mathématique des règles de travail.

Question 3 *Classifier les règles de travail de* (NSP_1) *selon leur type en précisant, selon les cas, les activités et les plages de temps considérées. Décrire les règles de séquencement par des expressions logiques.*

Question 4 *Parmi les règles de travail de* (NSP_1)*, lesquelles peuvent être facilement intégrées dans la représentation graphique discutée ci-avant ?*

C.3 Modèle de satisfaction de contraintes avec Choco.

Question 5 *Proposer un modèle CSP de* (NSP0)*, au moyen des contraintes disponibles dans Choco.*
`http://www.emn.fr/z-info/choco-solver/choco-documentation.html`
`http://www.emn.fr/z-info/choco-solver/choco-kernel/apidocs/index.html`

Question 6 *Proposer un modèle CSP des règles de travail de* (NSP_1)*, au moyen des contraintes disponibles dans Choco.*

C.4 Implémentation.

Un début d'implémentation de votre prototype vous est proposé. Il comprend les 5 classes suivantes :

- **NSData** : enregistre les données d'instance ; fournit les accesseurs sur les données (`nbDays()`, `nbEmployees()`, `getMinActivityCover(int a)`, `getMinWork(int e)`, `getMaxWorkPerWeek(int e)`, ...) et un constructeur pour l'instance (`makeInstanceNSP1()`);
- **NSChecker** : vérifie la réalisabilité d'une solution vis à vis des données d'instance ; fournit la méthode de vérification (`checkSolution(int[][] sol)`);
- **NSModel** dérivée de `CPModel` : modèle Choco spécialisé pour le NSP ; le constructeur appelle la création des variables (`makeVariable()`); fournit les méthodes de lecture de la solution calculée par le solveur, `solutionToString(Solver s)` pour l'affichage et `solutionToArray(Solver s)` pour le checker ;
- **NSModelConstrained** dérivée de `NSModel` : créée les contraintes Choco et les ajoute au modèle *méthodes à implémenter* ;
- **NurseScheduling** : classe principale ; fournit la méthode `testNSP()`); qui construit l'instance, le modèle, le solveur, lance la résolution vérifie ; et affiche la solution.

Dans un premier temps (modélisation), seule la classe `NSModelConstrained` devra être complétée.

Question 7 *D'ici la prochaine séance :*

- *téléchargez le code sur Campus et importez le projet sous Idea IntelliJ ou sous Eclipse ;*
- *générez la javadoc et exécutez* `NurseScheduling` *;*
- *implémentez la classe* `NSModelConstrained`*, avec les contraintes du modèle Choco pour l'instance* (NSP_1).

D Séance 2 : Améliorer le filtrage.

D.1 Contraintes redondantes et conjonction de contraintes.

Question 8 *Montrer que le réseau de contraintes suivant est* arc-cohérent *et* 2-cohérent *mais qu'il n'est pas* cohérent.

$$(x_1, x_2, x_3) \mid x_1 \neq x_2, x_2 \neq x_3, x_3 \neq x_1,\ x_1, x_2, x_3 \in \{0, 1\}$$

Montrer que le modèle suivant est cohérent *et* 2-cohérent *mais pas* arc-cohérent (généralisé).

$$(x_1, x_2, x_3) \mid \texttt{alldifferent}(x_1, x_2, x_3),\ x_1, x_2, x_3 \in \{0, 1\}, x_3 \in \{0, 1, 2\}$$

Quel est l'intérêt de la contrainte globale `alldifferent` *comparée à la conjonction de contraintes d'inégalités ?*

Améliorer un modèle de contraintes consiste à augmenter sa capacité de filtrage. Le coût algorithmique supplémentaire fourni à chaque nœud de l'arbre de recherche doit alors être contre-balancé par une forte réduction de la taille de cet arbre. Pour augmenter le filtrage, il existe principalement trois méthodes :

- changer l'algorithme soujacent à une contrainte globale : des contraintes telles que `alldifferent` sont disponibles dans Choco avec différents degrés de filtrage (cohérence de borne ou cohérence d'arc) ;
- ajouter des *contraintes couplantes* modélisant les interactions entre différentes contraintes du modèle portant sur un même ensemble de variables ; ces contraintes sont dite *redondantes* car elles ne sont pas nécessaires à la modélisation du problème ;
- combiner (et remplacer) plusieurs contraintes en une, modélisant chaque contrainte indépendamment ainsi que leurs interactions.

Question 9 *Identifiez et implémentez une contrainte couplante entre l'ensemble les contraintes de couverture et l'ensemble des contraintes de cardinalité.*

Par la suite, nous nous intéressons à la combinaison de l'ensemble des règles de séquencement d'un employé en une unique contrainte globale.

D.2 Langage des séquencements interdits.

Définition 1 (Alphabets et Mots) *Un* alphabet Σ *est un ensemble de symboles et un* mot *sur l'alphabet est une séquence de ses symboles. Une sous-séquence* $m_i m_{i+1} \dots m_k$ *d'un mot* $m = m_0 m_1 \dots m_l$ *est appelé un* motif apparaîssant *dans le mot* m *à la position* i. *Tout sous-ensemble de mots sur* Σ *forme un* langage.

L'emploi du temps d'un employé correspond de manière unique à un mot de longueur 28 sur un certain alphabet $\Sigma = \{D, N, R\}$: le i-ème symbole du mot dénote l'activité affectée à l'employé au i-ème jour. L'ensemble des emploi du temps possibles pour un employé e forme donc un langage L^e sur Σ et toute règle de travail se traduit par une restriction de ce langage. En particulier, toute règle de séquencement se traduit par un ensemble de motifs interdits, qui ne peuvent apparaître à certaines ou aucunes positions, dans un mot de L^e.

Question 10 *Déterminez les motifs interdits associés à chacune des règle de séquencement de* (NSP_1) *et précisez les positions interdites. Pour plus de concision, on introduit les symboles* W *(correspondant à toute activité travaillée* D *ou* N*) et* A *(correspondant à toute activité* D *ou* N *ou* R*) dans le langage des motifs ; ce qui permet de regrouper plusieurs motifs en un seul, par exemple :* $WDA \equiv \{DDD, DDN, DDR, NDD, NDN, NDR\}$.

Langages rationnels. L'*union* de deux langages, notée $L = L_1|L_2$, est l'ensemble des mots de L_1 et de L_2. La *concaténation* de deux langages, notée $L = L_1L_2$, est l'ensemble des mots formés par concaténation d'un mot de L_1, suivi par un mot de L_2. La *fermeture de Kleene* d'un langage L, notée $L*$ est définie récursivement comme l'union du mot vide et de l'ensemble des mots formés par concaténation d'un mot de L et d'un mot de $L*$. Le *complément* du langage L sur Σ est l'ensemble $\neg L$ des mots de $\Sigma*$ qui n'appartiennent pas à L. Un langage est *rationnel* s'il vérifie l'une des conditions suivantes :

- il est vide ;

– il contient un unique mot $a \in \Sigma$ (il est alors lui-même noté a) ;
– il est l'union de deux langages rationnels ;
– il est la concaténation de deux langages rationnels ;
– il est la fermeture de Kleene d'un langage rationnel.

Étant obtenu par combinaison au moyen des opérateurs $L_1L_2, L_1|L_2, L*$, tout langage rationnel peut être décrit par une chaîne de caractères, appelée expression rationnelle, qui répond à cette syntaxe de base sur l'ensemble des symboles de Σ et munie des caractères associatifs $()$. Pour simplifier l'écriture, nous étendons ici la syntaxe en introduisant l'opérateur redondant $L\{n\}$ qui dénote le langage L concaténé n fois, par exemple : $L\{7\}$ pour LLLLLLL. [1] Par exemple, le langage correspondant à l'expression rationnelle $(D|N|R)\{3\}(N|D)(N|D)R(D|N|R)*$ est l'ensemble des mots dans lesquels apparaît le motif *WWR* à la position 3.

Question 11 *Décrivez, dans cette syntaxe, le langage rationnel L_i des mots de L^e interdits par la règle de séquencement (i) pour tout $i = 1, \ldots, 5$. Ces langages étant donnés, décrivez formellement le langage L^e_s correspondant à l'ensemble des emplois du temps pour un employé qui respectent les règles de séquencement de (NSP_1).*

Automates finis déterministes. Un *automate* est un multi-graphe orienté valué (E, T, Σ) dans lequel on distingue deux sous-ensembles particuliers de sommets $S \subseteq E$ et $T \subseteq E$. Les sommets E sont appelés les *états*, les sommets de S les *états initiaux* et les sommets de T, les *états acceptants* ou *finaux*. Les arcs T sont appelés des *transitions* dont les valeurs sont des éléments d'un certain *alphabet* Σ. Un mot sur l'alphabet Σ est *reconnu* par l'automate s'il existe un chemin de S vers T tel que la séquence des valeurs des arcs du chemin forme le mot. L'ensemble des mots reconnus par l'automate forme le *langage reconnu* par l'automate.

Une classe particulière d'automates est la classe des automates déterministes finis. Un automate est *déterministe fini* si :
– les ensembles E et T sont finis ;
– il existe un unique état initial $S = \{s\}$;
– il n'existe pas deux transitions de même valeur issues du même état.

Théorème 1 (Théorème de Kleene) *Un langage est rationnel si et seulement s'il est reconnu par un automate déterministe fini.*

Par exemple, le langage défini par l'expression régulière $(D|N|R)*\,N(D|(R(D|N)))(D|N|R)*$ est le langage reconnu par les deux automates finis ci-dessous. Seul l'automate de droite est déterministe.

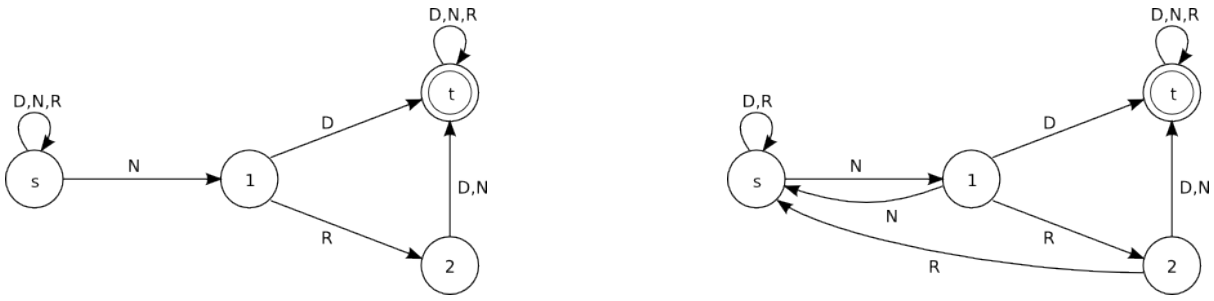

FIGURE 1 – Deux automates finis reconnaissant le langage $(D|N|R)*\,N(D|(R(D|N)))(D|N|R)*$ avec 4 états dont un état initial s et un état final t. (Les transitions mutiples entre deux états sont représenté par un seul arc avec plusieurs valeurs.)

Toutes les opérations (concaténation, union, complément) sur les langages sont applicables aux automates finis correspondants, auxquelles s'ajoutent l'opération de *déterminisation* qui transforme un automate fini non-déterministe en automate fini déterministe reconnaissant le même langage, et l'opération de *minimisation* qui transforme un automate fini en un automate fini reconnaissant le même langage et dont le nombre d'état est minimal.

1. Remarque : d'autres syntaxes sont possibles. Nous utilisons celle-ci qui est reconnue dans Choco.

D.3 Implémentation avec la contrainte `regular` de Choco.

Étant donnés une séquence finie de variables entières $x = (x_1, \ldots, x_n)$ et un langage rationnel L sur l'alphabet des nombres naturels, la contrainte `regular`(x, L) est satisfaite si et seulement si la séquence des valeurs $x_1 x_2 \ldots x_n$ forme un mot appartenant au langage L.

La contrainte globale `regular` est implémentée dans Choco et est accessible par l'API :

```
regular(IntegerVariable[] x, FiniteAutomaton fa)
```

où `fa` est un automate fini reconnaissant le langage auquel le mot `x` doit appartenir. L'algorithme soujacent assure la cohérence (généralisée) d'arc en un temps linéaire en la taille de `x` et la taille de `fa`.

La classe FiniteAutomaton de la librairie Choco offre l'API suivante pour construire et manipuler des automates finis à valeurs dans l'ensemble des nombres naturels (plus exactement, l'ensemble des entiers positifs inférieurs à Character.MAX_VALUE) :

```
FiniteAutomaton();
FiniteAutomaton(String regExp);

int addState();
void setInitialState (int state );
void setFinal(int state );
void addTransition(int stateFrom, int stateTo, int ... symbols);
void addToAlphabet(int symbol);

FiniteAutomaton union(FiniteAutomaton a);
FiniteAutomaton intersection(FiniteAutomaton a);
FiniteAutomaton complement();

void minimize();

int getNbStates();
void toDotty(String dotFileName);
```

Le second constructeur, notamment, prend en argument une expression régulière regExp répondant à la syntaxe décrite ci-avant, avec les opérateurs de concaténation, d'union '|', d'association '(' et ')', et les quantificateurs infini '$*$' et fini '{n}'. De plus, les valeurs entières supérieures à 10 doivent être spécifiées entre chevrons, ex : <10> pour ne pas confondre avec la concaténation de 1 et de 0. L'opération complement retourne le complément de l'automate dans son alphabet. Par défaut, il s'agit de l'ensemble des valeurs des transitions de l'automate. Pour obtenir le complément dans un ensemble plus grand, il est nécessaire d'ajouter explicitement les symboles manquants dans l'alphabet de l'automate au moyen de addToAlphabet.

Question 12 *Dans votre modèle Choco de* (NSP_1)*, ajoutez (et non modifiez !) la modélisation des règles de séquencement au moyen de la contrainte* `regular`*.*

Question 13 *Quel modèle est le plus performant ? Le modèle avec les contraintes logiques, le modèle avec une contrainte* `regular` *pour chaque règle de séquencement, ou le modèle avec une unique contrainte* `regular` *pour toutes les règles de séquencement ?*

Question 14 (*) *Implémentez un dernier modèle basé sur la contrainte* `multiCostRegular` *conjugant l'ensemble des contraintes de travail. (Voir l'exemple de la documentation.)*

E Séance 3 : Améliorer la recherche.

E.1 Stratégies de recherche.

Nous avons vu précédement comment améliorer le modèle en ajoutant des contraintes redondantes et en combinant différentes contraintes en une seule pour augmenter la propagation. Dans cette séance, nous allons jouer avec la manière dont l'arbre de recherche est créé et parcouru, en testant diverses heuristiques de choix de variables et de valeurs pré-implémentées dans la librairie Choco.

La section *Search strategy* de la documentation de Choco décrit les heuristiques disponibles. Nombre d'entre elles sont accessible par la classe de construction `BranchingFactory`.

Question 15 *Quelles sont les variables de décision du modèle ? Montrer que l'instanciation complète de ces variables entraîne l'instanciation de toutes les variables du modèle.*

Question 16 *Quelles sont les heuristiques par défaut du solveur de Choco ? Étudiez les autres heuristiques disponibles.*

Question 17 *Implémentez et testez différentes heuristiques de branchement sur les variables de décision.*

E.2 Symétries.

Un modèle mathématique présente une *symétrie* s'il existe une relation d'équivalence sur l'ensemble de ses solutions, par exemple, si plusieurs solutions *du modèle* correspondent à une même solution *du problème*, ou à des solutions semblables qui peuvent être facilement construites les unes à partir de toute autre. De nombreux problèmes et modèles possèdent des symmétries évidentes. Casser ces symmétries consiste à n'autoriser qu'une seule solution par classe d'équivalence. Il s'agit par conséquence, d'ajouter des contraintes au modèle, qui ne sont pas des contraintes du problème initial, afin de réduire l'espace des solutions, et donc l'arbre de recherche.

Question 18 *Exhibez des* symétries *du modèle et modélisez.*

F Séance 4 : Bilan et analyse.

Question 19 *Modularisez le code de façon à tester différentes combinaisons de modèles/heuristiques pour le problème NSP1. Enregistrez les résultats d'exécution (temps de résolution, nombre de nœuds et de backtracks) dans un fichier. Préparez une table de résultats comparatifs des différentes combinaisons testées, éventuellement des graphiques, et une analyse, le tout consigné dans un rapport. Finalisez le code (javadoc notamment).*

Question 20 *Renommez le package principal de votre code à votre nom :* `nurseSchedulingDupont`. *Préparez une archive* `nsDupont.zip` *contenant uniquement le répertoire des sources* `nurseSchedulingDupont` *et votre rapport* `nsDupont.pdf`. *Déposez sur Campus.*