TP 2 : SATO

14 novembre 2006

Le but de ce TP est de résoudre un problème d'emploi du temps, en le traduisant vers un problème de satisfiabilité. On utilisera ensuite un solveur pour résoudre ce problème de satisfiabilité.

1 Un problème d'emploi du temps

Brian dirige un centre de vacances depuis de nombreuses années. En raison du succès grandissant de ses nombreuses activités, il est confronté à un public de plus en plus nombreux, et il lui est difficile de s'organiser pour éviter les recouvrements de cours : si un des participants est inscrit à la fois au cours de kyte-surf et de body-building, il faut éviter que les deux cours aient lieu en même temps !

Heureusement, Brian peut faire appel à son ami Bobby le surfeur, qui possède une licence d'informatique. Bobby propose fort naturellement de réduire le problème à une question de satisfiabilité de clauses. Il considère l'ensemble des activités proposées $\{a_1, \ldots, a_n\}$ et des créneaux horaires disponibles $\{h_1, \ldots, h_p\}$ (typiquement, les jours de la semaine). Les variables propositionnelles sont notées $x_{a,h}$: la variable $x_{a,h}$ vaudra `true` si l'activité a a lieu à l'horaire h, sinon elle vaudra `false`.

À chaque participant p est associé un ensemble $\mathcal{A}(p)$ d'activités. On dit que deux activités distinctes a et a' sont connectées s'il existe un participant p tel que $a, a' \in \mathcal{A}(p)$.

Les contraintes à imposer sont de trois types :

(1) si deux activités distinctes sont connectées, alors les deux activités ne peuvent avoir lieu au même créneau horaire

(2) chaque cours doit avoir lieu à au moins un créneau horaire

(3) chaque cours ne peut avoir lieu à deux créneaux horaires distincts.

Exercice 1 *Écrire les clauses correspondant à chaque type de contraintes. On précisera bien sur quel ensemble varient les paramètres de la clause.*

Exercice 2 *Cindy, la petite sœur de Bobby, fait alors remarquer que les contraintes du type **(3)** ne sont pas utiles à la résolution du problème. Pourquoi ? Comment trouver une solution respectant les contraintes **(1)**-**(2)**-**(3)** à partir d'une solution respectant les contraintes **(1)**-**(2)** ?*

Par la suite, on considérera uniquement les clauses correspondant aux conditions **(1)** et **(2)**. On va utiliser le solveur SATO pour chercher une solution à cet ensemble de clauses.

2 Le solveur SATO

Le programme SATO est un solveur qui prend comme entrée une formule CNF dans un fichier sous forme DIMACS et retourne, si elle existe, une affectation qui satisfait cette formule.

Avant de commencer, vous devez installer ce programme sur votre compte. Pour cela, vous devez :

– récupérer le fichier `sato4.2.tgz` à l'adresse suivante :
`http ://www.cs.uiowa.edu/~hzhang/sato/`
– décompresser l'archive en exécutant la commande :
`> tar xvf sato4.2.tgz`
– compiler le programme en exécutant dans le dossier de SATO la commande :
`> make`

Plus d'informations sont disponibles dans le fichier `README` à la même adresse.

Vous pouvez tester le solveur en utilisant par exemple le générateur programmé dans le TP1 et comparer les performances avec votre implémentation de l'algorithme de Davis-Putnam.

3 Interface

La principale tâche à effectuer est l'interfaçage entre le système de clauses et SATO. Dans un premier temps on doit générer le système de clauses à satisfaire. Puis on doit transformer une affectation qui satisfait la formule générée en un emploi du temps.

Notez qu'en Java, on ne peut pas faire d'appel système, il faudra donc exécuter le programme final à l'aide d'un script de la forme :

```
java EdT2SAT fic_entree
./sato fic_dimacs > fic_sol
java SAT2EdT fic_sol
```

où `fic_entree` est le fichier qui représente les données en entrée (i.e. les desiderata des participants), `fic_dimacs` est la traduction du problème en format DIMACS produit par le programme `EdT2SAT`, et `SAT2EdT` traduit l'affectation retournée par SATO en solution du problème initial. Le but du TP est de programmer `EdT2SAT` et `SAT2EdT`.

4 EdT → SATO

Le fichier d'entrée qui représente les desiderata des participants aura le format suivant :

```
edt 9 4 5
4 5 3 7 6
8 9 7 1
5
4 7 1
```

La première ligne indique qu'il y a 9 activités, 4 participants et 5 créneaux horaires. La deuxième ligne indique que le premier participant désire assister aux activités `4`, `5`, `3`, `7` et `6`, et ainsi de suite pour les autres lignes.

Exercice 3 *Écrire un programme qui prend en entrée un tel fichier, et qui retourne dans un fichier de sortie la formule CNF correspondant aux contraintes d'emploi du temps, sous le format DIMACS.*

5 SATO → EdT

Il reste à transformer une affectation renvoyée par le solveur en un emploi du temps respectant les contraintes.

Exercice 4 *Écrire, une fonction qui transforme une affectation qui satisfait la formule en un emploi du temps effectif. On veillera à respecter dans ce cas la condition **(3)**. Tester le programme de génération d'emploi du temps sur des exemples simples.*

6 Tests : le principe du pigeonnier

Pour tester le programme et ses performances, on se propose de revenir au problème du pigeonnier (cf. TP1). En effet, si on a m pigeons et n nids, la situation se ramène à un problème d'emploi du temps avec m activités et n créneaux horaires, où toutes les activités sont connectées (par exemple, on a un seul participant qui veut suivre toutes les activités).

Exercice 5 *Programmer une fonction qui génère le fichier d'entrée correspondant au problème du pigeonnier à partir de la donnée de n et m. Tester les performances du solveur pour $n = m$ et $n = m - 1$: jusqu'à quelles valeurs de m ces deux cas sont-ils résolus ? Comparer à la version qui utilise Davis-Putnam.*