Projet MIM2 - Rapport de mi-projet

Laure Danthony et Vivian Brégier

Pour le 6 novembre 2001

Réalisation d'un logiciel d'emploi du temps

Introduction

L'objectif de ce projet est de réaliser un logiciel graphique et simple d'emploi dont le but est d'aider Simone Durand lors de la réalisation semestrielle des emplois du temps du MIM. Ce logiciel devra fonctionner sous Windows et les détails d'implémentation ne devront en aucun cas transparaître. Les contraintes imposées pour l'emploi du temps sont celles effectivement rencontrées par Simone.

1 Cahier des charges

Les contraintes retenues dans un premier temps sont du type :

1. créneaux autorisés (durée : 2h). On pourra ainsi décider que le jeudi après-midi ne possède pas de créneau pour cause de sport.
2. disponibilités des professeurs. Tel professeur faisant tel cours peut n'être disponible que le mercredi, par exemple.
3. intersection de cours des élèves. Deux cours ne peuvent être parallélisés (*i.e.* avoir lieu en même temps) si un seul élève suit ces deux cours.

Pour l'instant, on fait le choix de ne pas tenir compte des salles disponibles, ni de la possibilité de dédoubler certains TD pour cause de sur-effectif.

2 Implémentation

2.1 Vocabulaire

On utilisera le vocabulaire suivant dans la suite et pour l'implémentation :

- Un **cours** est indifféremment un cours proprement dit ou un td, sa seule caractéristique est qu'il occupe un créneau de 2 heures.
- Un **créneau** est une plage horaire du type $(jour, debut)$. Ainsi $(lundi, 10)$ est un créneau. Pour une journée donnée, une plage horaire appartient à l'ensemble $\{8, 10, 14, 16\}$
- Une **liste de cours** d'élèves est une liste d'intitulés de **cours** : pour un élève donné, une telle liste est de longueur deux fois le nombre de modules choisis (on compte les cours et les td).

2.2 Choix des structures de données

On utilisera les structures de données suivantes :

1. Un `cours` sera un type "record" contenant les informations : le nom du cours, le nom du professeur, les élèves suivant le cours, la liste des créneaux possibles, et un champ "souhait" qui pourra éventuellement servir plus tard pour affiner "à la main" ;
2. Une structure de données `eleves` contiendra pour chaque élève son nom, la liste des modules pris et un champ "souhait" qui pourra éventuellement servir plus tard ;
3. Un graphe d'interdictions `graph_depend` où deux sommets de type `nom_de_modules` seront reliés ssi il existe un élève qui prend ces deux modules. Il s'agira d'une donnée globale, calculée au début une fois pour toutes. (ce n'est pas difficile à faire).
4. Une `situation` (en cours) contiendra à tout moment de l'algorithme d'allocation :
   - les modules pour lesquelles ont a déjà attribué un créneau,
   - les modules qui restent à placer,
   - pour chaque créneau de la semaine, les modules qui peuvent encore être mis dans ce créneau,
   - les plages encore possibles pour chaque module.

   Les deux dernières informations sont redondantes mais on ne peut se permettre de renverser très souvent des tables, étant donné la complexité de l'algorithme. Cette structure est la strucure clef de l'algorithme, celle qui sera constamment utilisée.

2.3 Remarques sur la stratégie d'allocation

L'ordre dans lesquels les cours devront être alloués est très important, on doit commencer par les contraintes des professeurs les plus exigeants pour scier au plus tôt "les branches mortes" de l'arbre de parcours des possibilités. Il faut bien remarquer qu'à chaque étape, il faut recalculer ces priorités de traitement car par exemple YR qui n'était pas très exigeant au début (plages possibles : tout sauf le lundi et le vendredi), devra être placé prioritairement dès que toutes les plages du mardi, mercredi et jeudi seront allouées !

2.4 Un premier algorithme

Un premier algorithme réalisera les étapes suivantes :

- **Initialisation :** initialisation des structures, calcul du graphe d'interdiction, calcul des priorités initiales pour les cours, calcul des cours pouvant être parallélisés ...En particulier, on initialise la structure `situation` bien soigneusement.
- **Boucle** : On désire traiter une situation s. On choisit pour cela un cours M le plus restrictif possible et non encore alloué. Pour chacun des horaires C possibles, on va essayer d'attribuer cet horaire au cours, en faisant une copie de la situation,dans laquelle on change la liste des cours à attribuer, les cours de chaque créneau ...et essayer d'appeler récursivement le programme sur cette nouvelle situation. Si cet appel récursif n'aboutit pas (*i.e.* lance une exception `Echec`), on essaie d'attribuer un autre

horaire. Sinon, on reçoit de cet appel récursif un ensemble Γde couples (cours,horaire) et on renvoie l'ensemble $\Gamma \cup \{(M, C)\}$. Si aucun horaire n'est possible, on lève l'exception `Echec`. Cela signifie que la situation s n'admet pas de solution.

3 Etat d'avancement

Les choix d'implémentation étant faits, il n'y a "plus qu'à" coder. On choisit le langage OCAML, nous interfacerons ensuite avec Java ou Camltk pour réaliser une jolie application Windows. Il faudra pour cela entre autres faire le choix d'un langage pour coder les différentes contraintes.

Prochaines étapes :
- Code des structures de données et de l'algorithme.
- Tests sur des exemples réels fournis par Simone.