Algorithmes – Emploi du temps

Coudray – Julien – Tran

18 mars 2014

1 Pré-traitements

Avant de réaliser l'emploi du temps, le programme procède à des vérifications sur les données d'entrées afin de détecter toutes les incohérences. Ainsi, il élimine au préalable une partie des traitements qui n'aboutiront pas.

1.1 Le nombre de professeur

La première vérification concerne le nombre de professeurs en entrée. Nous vérifions s'il y a assez de professeurs pour dispenser les cours de chaque classe. Ainsi pour un cours donné, l'algorithme somme les disponibilités des professeurs puis compare le résultat au nombre de classe.
Si le cours est sur 4 heures, alors la somme des disponibilités est divisée par 2. En effet, le cours en question nécessite deux créneaux consécutifs pour être dispensé.

Soit n le nombre de professeurs pouvant donner un cours c et m le nombre de classe devant suivre ce cours. Pour un cours de 2 heures, nous avons :

$$\sum_{i=0}^{n} dispo_{prof_i} > m$$

Pour un cours de 4 heures, nous avons :

$$\frac{\sum_{i=0}^{n} dispo_{prof_i}}{2} > m$$

L'opération est répétée pour l'ensemble des cours.

Algorithme 1 : Pré-traitement nombre de professeurs

```
for all Courses do
    idCourse ← identifiant de Courses
    idPromo ← identifiant de la promotion recevant Courses
    nbClasses ← nombre de classe de la promotion idPromo
    for all Profs do
        if Profs donne le cours idCourse then
            for all CreneauxProf do
                if Profs est disponible then
                    nbCreneaux ← nbCreneaux + 1
                end if
            end for
        end if
    end for
    if Courses est sur 4h then
        nbCreneaux ← nbCreneaux/2
    end if
    if nbClasses > nbCreneaux then
        display (Erreur sur le nombre de professeur pour la promo idPromo)
        EXIT FAILURE
    end if
end for
display (Nombre de professeurs ok)
```

1.2 Le nombre de cours total sur le semestre

La seconde vérification porte sur le nombre d'heures de cours à dispenser à une classe. Ce nombre ne doit pas excéder la totalité des heures du semestre. Le programme somme l'ensemble des cours que possède une classe et le compare au nombre d'heures du semestre.

Soit n le nombre de cours d'une classe p, s le nombre de semaines sur un semestre, c le nombre de créneaux sur une semaine et h le nombre d'heures d'un créneau :

$$\sum_{i=0}^{n} nbHeures_{cours_i} \leq s * c * h$$

Algorithme 2 : Pré-traitement nombre d'heures sur le semestre

```
for all Classes do
    listCourses ← ensemble des cours que suit une classe
    for all courses in listCourses do
        nbHours ← nbHours+ nombre d'heures du cours courses
    end for
    if nbHours > (nombre de semaines du semestre * nombre de créneaux par semaine *
    nombre d'heures par créneau) then
        display(Erreur, trop d'heures pour la classe Classes)
        EXIT FAILURE
    end if
end for
display(Nombre d'heures de cours ok)
```

Une fois ces pré-traitements réalisés, nous pouvons commencer la conception de l'emploi du temps de l'école.

2 Réalisation de l'emploi du temps

La réalisation de l'emploi du temps se déroule en deux étapes qui sont réalisées indépendamment pour chaque promotion. Chaque classe d'une promotion suit le même programme avec la même liste d'enseignants potentiels. C'est pourquoi chaque promotion va avoir un emploi du temps sur le semestre. A savoir, la planification des cours indiquant la semaine de début et de fin. Enfin, un emploi du temps final sera réalisé semaine par semaine avec chaque cours placés sur ses créneaux respectifs.

Algorithme 3 : Principe général de conception des emplois du temps

for all *Promo* **do**
  $idCourses \leftarrow$ liste de tous les cours que doivent suivre la promotion *Promo*
  repartitionCoursSemestre($idCourses$, $programmeSemestre$)
  repartitionCoursPromotions(*Promo*, $programmeSemestre$)
end for

2.1 Répartition du programme sur le semestre

La première étape de l'algorithme de résolution est de répartir de l'ensemble du programme de chaque promotion sur le semestre. Cette étape consiste à indiquer pour chaque cours la date de début et de fin semaine.

La répartition se déroule en deux étapes :
— Le trie des cours
— Le placement des cours sur le semestre

L'objectif est de répartir au mieux les cours sur le semestre. Il faut donc réussir à placer le maximum de cours les uns à la suite des autres. C'est pourquoi nous plaçons les cours les plus longs en premier, puis nous vérifions s'il est possible de placer un nouveau cours derrière ceux-là, sinon nous le plaçons en début de semestre.

Un cours de 4 heures impose plus de contraintes. En effet, il s'agit d'un cours où le professeur et la classe doivent avoir deux créneaux consécutifs dans la même demi-journée. C'est pourquoi un cours de 4 heures doit être planifié sur le semestre avant un cours de 2 heures.

Pour se faire, les cours vont être séparés en deux listes : une pour les cours de 4 heures et une autre pour les cours de 2 heures. Ainsi pour chacunes des listes, un trie décroissant est effectué par rapport au nombre de semaines sur lequel les cours vont être suivis.

$$semaineDebut_{coursPlace} + nbSemaine_{coursPlace} + nbSemaine_{nouveauCours} \leq nbSemaine_{semestre}$$

Chaque élément du semestre va avoir les informations suivantes :
— L'identifiant du cours
— Le numéro du début de la semaine
— Le nombre de semaine du cours

— Le cours qui le suit

Algorithme 4 : Algorithme principale de la répartition des cours sur le semestre

Require: liste *idCourse*, liste *programmeSemestre*
  for all *idCourse* **do**
    if *idCourse* est un cours sur 2h **then**
      *idCourses*2 ← pushback *idCourses*
    else
      *idCourses*4 ← pushback *idCourses*
    end if
  end for
  Trie de *idCourses*2 par nombre de semaine de cours décroissant
  Trie de *idCourses*4 par nombre de semaine de cours décroissant
  idCourses est vider
  idCourses ← pushback *idCourses*4
  idCourses ← pushback *idCourses*2
  programmeSemestre ← repartitionDesCours(*idCourses*)

Algorithme 5 : repartitionDesCours($idCourses$)

Require: liste $idCourses$ triée par nombre de semaine d'un cours et par cours de 4H et 2H
initialisation de $coursProgrammes$
for all $idCourses$ **do**
  for all $coursProgrammes$ **do**
    if $programmeSemestre$ a cours placé **then**
      checkNextCourse($idCourses$, $coursProgrammes$)
      if $idCourses$ a été programmé **then**
        $coursPlace \leftarrow true$
        BREAK
      end if
    end if
  end for
  if $coursPlace == false$ **then**
    $coursProgrammes \leftarrow$ pushback $idCourses$ en début de semestre
  end if
end for
return $coursProgrammes$

Algorithme 6 : checkNextCourse($idCourses$, $coursProgrammes$)

if $coursProgrammes$ a un cours après lui déja **then**
  checkNextCourse($idCourses$, $coursProgrammes$ du cours suivant)
else if $semaineDebut_{coursProgrammes} + nbSemaine_{coursProgramme} + nbSemaine_{idCourses} \leq nbSemaine_{semestre}$ **then**
  $coursProgramme \leftarrow$ pushBack $idCourses$
end if