Intelligence Artificielle : Logique et Contraintes 2 - Devoir

Génération d'emplois du temps

Ce projet peut être réalisé en binôme.

Il devra être rendu directement dans votre projet sur la plateforme gitlab de l'université, dans le répertoire `src/opl/devoir`.

La date de rendu est fixée au **vendredi 28/04/23 à 23h59**

Vous veillerez à réaliser régulièrement des commit / push sur votre projet au fur et à mesure de l'avancement de votre travail et à mettre un message pertinent à chaque étape. Le commit final devra être associé au message "rendu final devoir".

La dernière séance du cours ne comprendra pas de sujet de TP mais sera une séance d'assistance à votre devoir. Il est donc dans votre intérêt d'essayer de commencer au plus tôt et de noter des à présent les questions que vous pourriez vous poser.

Si certains points vous semblent confus dans l'énoncé, n'hésitez pas à poser des précisions. Les réponses seront systématiquement postée sur la page du cours, afin que tous puissent en profiter.

Le but de ce devoir est de construire un système capable de construire des emplois du temps pour un établissement d'enseignement supérieur.

Nous considérons des cas où l'on veut générer des emplois du temps pour différentes filières, chaque filière ayant ses caractéristiques propres, mais pouvant éventuellement partager certains cours (comme par exemple en licence ou master). Un cas typique d'utilisation est de générer un emploi du temps pour période correspondant à une semaine type, qui sera reconduit de semaine en semaine durant un semestre. Mais pour certains établissements comme par exemple des écoles d'ingénieurs, qui ont beaucoup d'intervenants extérieurs avec de fortes contraintes de disponibilité, les emploi du temps peuvent varier suivant les semaines. On souhaite donc une solution souple permettant de générer des emploi en fonction d'une période de longueur (i.e. nombre de jours) variable.

Construire un emploi du temps consiste à déterminer, pour chaque cours d'un ensemble donné, le moment où pourra se dérouler ce cours et la salle où il aura lieu. Les salles disponibles étant généralement mutualisées entre les différentes formations, il est nécessaire de convenir de règles communes pouvoir caler les réservation de salles.

Dans la suite, nous supposerons qu'une journée est divisée en un certain nombre de créneaux horaires, de durées identiques, et que la réservation d'une salle pour un cours s'effectue toujours sur des créneaux entiers (même si l'horaire du cours fait qu'il ne couvre pas complètement un créneau. Le choix de la durée des créneaux horaires est donc un élément déterminant pour varier le niveau de granularité des problèmes considérés. Un cours d'une d'une duré de 120mn se déroulera sur deux créneaux horaires si la durée des créneaux est de 60mn, ou de 90mn, mais un seul si celle-ci est de 120mn. Nous supposerons aussi l'existence d'une pause méridienne (qui n'est pas un créneau en tant que tel), pendant laquelle aucun cours ne peut avoir lieu. Les cours doivent donc se dérouler intégralement avant ou après la pause méridienne.

1 Caractéristiques des instances

Nous supposerons qu'une instance de problème à résoudre est caractérisé par un ensemble d'informations qui sont détaillées par la suite.

- un nom, qui doit correspondre au chemin d'accès (relatif) depuis le répertoire de base (celui où se trouve vos fichiers .mod) jusqu'au fichier d'instance. Conseil : choisir autant que possible des noms de fichiers qui reflètent les caractéristiques de chaque instance

- le nombre de jours de l'emploi du temps
- le nombre de créneaux par jour
- la durée de chaque créneau (exprimée en minutes)
- l'horaire du premier creneau
- le numéro du créneau après lequel se déroule la pause méridienne
- la durée de la pause méridienne (exprimée en minutes)
- un ensemble des filières
- l'ensemble des parcours des différentes filières
- un ensemble de cours
- un ensemble de cours donnant lieu à plusieurs séances
- un ensemble de salles et leurs caractéristiques
- un ensemble d'indisponibilités pour les professeurs
- un ensemble d'indisponibilités pour les salles
- trois ensembles permettant de décrire d'éventuelles contraintes pour le positionnement de certains cours dans l'emploi du temps, comme le fait qu'un cours doit se dérouler avant un autre, ou en même temps qu'un autre, où encore pour forcer son placement dans l'emploi du temps à un moment précis.

Nous supposerons que chaque instance décrit toujours au moins une filière, avec au moins un parcours.

Nous supposerons les données caractérisant chaque instance sont déjà décrites dans la syntaxe du langage OPL (vous ferez en sorte de définir les structures adéquates permettant de lire de tels fichiers. sans modification). Un exemple de fichier décrivant une instance type est donné en annexe (cf. Annexe A), que nous utilisons pour préciser la sémantique du format de données.

2 Précisions sur le format des données et leur interprétation

L'objectif est de pouvoir générer des emploi du temps pour un ensemble de **filières**.

2.1 Filières et parcours

Une filière est caractérisée par un identifiant, un ensemble d'identifiants de cours et un ensemble d'identifiants de parcours. Les parcours permettent de préciser les variations d'emploi du temps en fonction des groupes auxquels ils sont rattachés. Les cours décrits au niveau de la filière sont communs à tous les parcours de cette filière.

Chaque parcours est caractérisé par un identifiant de parcours, le nombre d'étudiants concernés ainsi que l'ensemble des identifiants de cours spécifiques à ce parcours (qui s'ajoutent donc aux cours communs) . Si tous les étudiants d'une même filière suivent exactement les mêmes cours... il faut malgré tout déclarer un parcours, qui précisera l'effectif de la filière, mais dont l'ensemble des cours spécifiques pourra rester vide.

La notion de parcours est utile lorsqu'une promotion peut être découpée en différent groupes (e.g. pour les TDs et TPs) où encore lorsque certains cours sont optionnels, il peut y avoir dans ce cas de nombreux parcours différents pour une même filière. Tous les étudiants d'un même parcours suivent exactement les mêmes cours.

Exemple : L'exemple ci dessous décrit que la filière L3-Miage, comporte trois cours communs et 3 parcours distincts.

```
<"L3-Miage",  {"LogiqueL3","EcoL3IG","IG-Conf1"}, {"L3IGP1", "L3IGP2", "L3IGP3"}>
```

les parcours étant caractérisés par :

```
<"L3IGP1",  20,   {"EcoL3IG-TD1", "LogL3-TDIG1"} >,
<"L3IGP2",  10,   {"EcoL3IG-TD2", "LogL3-TD1"}>,
<"L3IGP3",  8,    {"EcoL3IG-TD2", "LogL3-TD2"}>
```

Notons que les étudiants des parcours "L3IGP2", "L3IGP3", sont regroupés dans un même groupe de TD en économie : "EcoL3IG-TD2" mais sont dans des groupes distincts pour les TDs de logique : "LogL3-TD1" et "LogL3-TD2".

2.2 Cours et séries

Un **cours** décrit un élément d'enseignement caractérisé par un identifiant (unique), une description de l'intitulé du cours, l'identifiant du professeur qui assure ce cours, la durée de ce cours (exprimée en mn) et les besoins spécifiques pour ce cours à prendre en compte lors de l'attribution des salles. Notons que si une matière est dispensé sous différentes formes (eg. cours magistral, td, tp) cela donnera lieu à différents cours. S'il est nécessaire de répartir les étudiants d'une filière dans différents groupes, chaque groupe sera également considéré comme un cours distinct.

Exemple :

```
<"PFA",  "Programmation Fonctionnelle Avancée - Cours" "TomP",90,  {"video", "amphi"}>,
<"PFA-TP1",  "Programmation Fonctionnelle Avancée - TP1" "TomP",120,  {"tp_info"}>,
<"PFA-TP2",  "Programmation Fonctionnelle Avancée - TP21" "JeanF",120,  {"tp_info"}>,
```

permet de décrire un enseignement composé d'un cours magistral de 1h30, qui doit se dérouler dans un amphi disposant d'un vidéo-projecteur, associé à deux groupes de TPs qui durent 2h et doivent se dérouler dans des salles de tp informatique. C'est de plus le même professeur qui assure le cours et le groupe de TP1.

La notion de série est utile pour décrire des situations où un même cours peut donner à plusieurs séances durant la période considérée (par exemple un cours qui a lieu 2 fois par semaine, ou lorsque l'on veut générer un emploi du temps sur plusieurs semaines. Chaque série décrit alors le cours concerné (par son identifiant), le nombre total de séances à prévoir dans la période et le nombre minimal/maximal de jours d'écart entre 2 séances consécutives pour ce cours.

Exemple : La série suivante :

```
<"PFA",  3, 5, 5>,
```

décrit que le cours PFA a lieu 3 fois durant la période considérée et que deux séances consécutives doivent se dérouler exactement à 5 jours d'intervalle. Tout cours mentionné dans la description des filières et des parcours donne par défaut au moins lieu à 1 séance. Si aucun cours ne donne lieu à plusieurs séances dans la période considérée, l'ensemble des séries peut rester vide.

2.3 Salles

Chaque salle est décrite par un identifiant, une capacité d'accueil et une liste de "services" offerts par cette salle. La notion de service est ici à prendre au sens large. Par exemple :

```
<"PUIO-GrandAmphi", 250, {"video", "amphi", "cours"}>,
```

permet de décrire le grand amphi du PUIO, qui a une capacité d'accueil de 250 places, possède un video-projecteur, a la particularité d'être un amphi et plus généralement, d'être une salle possible pour donner des cours.

Si l'on veut se donner la possibilité de le choix d'une salle pour certains cours on peut par exemple rajouter l'identifiant de la salle comme un service, qui sera requis spécifiquement pour ces cours.

2.4 Indisponibilités

Pour tenir compte de contraintes extérieures, on veut pouvoir d'écrire que certains enseignants ne sont pas disponibles à certains moments. Chaque indisponibilité est caractérisée par l'identifiant du professeur, le numéro dans la période du jour d'indisponibilité (le premier jour étant le jour 1) et la fourchette horaire durant laquelle le professeur est indisponible. Par exemple :

```
<"JeanF",  7,  <7,00>, <14,00> >,
```

Permet de déclarer que le professeur JeanF, n'est pas disponible le jour 7h à 14h.

De la même façon on peut déclarer que certaines salles sont indisponibles à certain moments en utilisant un identifiant de salle à la place d'un identifiant de professeur.

Prendre en compte ces indisponibilités revient à s'assurer qu'un cours assuré par un professeur ne peut être placé (res. une salle ne peut être utilisée pour un cours) sur un créneau qui recouvre (même partiellement) la plage horaire d'indisponibilité.

2.5 Contraintes de placement sur les cours

Parfois on peut avoir envie de préciser certaines contraintes au niveau du placement de certains cours.

L'ensemble **precedences**, permet d'indiquer que certains cours doivent forcément avoir lieu avant d'autres. Par exemple :

```
precedences = {
  <"PFA",  "PFA-TP1">,
...
```

indique que le TP de PFA a lieu forcément après le cours de PFA. Lorsqu'un cours comprend plusieurs séances, cette contrainte doit s'entendre comme devant s'appliquer à chaque séance (i.e. la k-ieme séance de cours doit se produire avant la k-ieme séance de TP).

L'ensemble **memesHoraires**, permet d'indiquer que certains cours doivent forcément avoir lieu au même moment. Par exemple : xxxx

```
memesHoraires = {
<"PFA-TP1",    "PFA-TP2">
...
```

indique que les séances de TP de PFA des groupes 1 et 2 doivent se dérouler en parallèle. A nouveau, si plusieurs séances sont prévue, cela doit être le cas pour chaque séance.

Enfin l'ensemble **creneauxFixes**, permet de fixer jour et un créneau horaire pour une séance de cours particulière (si le cours nécessite plusieurs créneaux on considère qu'il s'agit du premier créneau sur lequel se déroule ce cours. Exemple :

```
creneauxFixes = {
<"IG-Conf1", 1, 2, 3>
};
```

permet d'indiquer que la première séance du cours `IG-Conf1` doit de dérouler (i.e. débuter) au troisième créneau horaire du second.

3 Modélisation du problème

3.1 Problèmes de satisfaction

Lors de la modélisation du problème, vous devrez identifier les différentes contraintes devant être respectées pour que l'emploi du temps soit pertinent. On se propose pour cela de procéder par étape on construisant :

1. Dans un premier de temps, un modèle `edt_simple.mod` permettant de construire un emploi du temps sans tenir compte des aspects liées aux salles.
2. Dans un second temps, un modèle `edt_salles.mod` qui permet en plus d'attribuer une salle à chaque cours, en respectant les besoins spécifiques et l'effectif de ce cours.

Remarque : pour mettre au point vos modèles il va vous falloir concevoir différents jeux d'instances. Il n'est pas nécessaire de que ces instances soient très compliquées. On peut proposer des instances sur très peu de jours et/ou très peu de parcours cours. Mais il faut vous assurer que vos instances couvrent

un peu tous les cas de figure possibles (faites attention notamment aux valeurs limites) et faites en sorte que vos modèles soient robustes par rapport aux contenu des fichiers d'instances.

En pratique vous regrouperez chaque instance permettant de faire de tels tests dans un répertoire nommé `data/tests` en prenant garde à donner un nom différent pour chaque instance.

Conseil : essayer de prévoir d'abord des instances capables de tester chaque caractéristique séparément... Par exemple, vous pouvez faire varier séparément le nombre de jours, le nombre de créneaux par jour, la durée des créneaux, l'horaire du premier créneau, de la pause méridienne, les caractéristiques des filières et des cours les indisponibilités, ...

Pour chaque instance de test, vous mettrez en tête du fichier .dat un commentaire précisant ce que l'instance permet de tester.

Ensuite vous pourrez concevoir de façon incrémentale des instances plus complexes permettant de tester simultanément plusieurs caractéristiques (avec plus ou moins de filières, de parcours, de salles, de contraintes de placement, ...). Essayez de comprendre comment les paramètres interagissent et ce qui peut rendre la résolution plus ou moins difficile (par exemple, la quantité et les caractéristiques des salles disponibles).

Considérez que la conception de jeux d'instances de test variés fait pleinement partie du travail a réaliser pour ce devoir.

3.2 Optimisation des emploi du temps

Une fois que vous serez assurés que les emploi de temps que vous générez satisfont bien toutes les contraintes que vous voulez prendre en compte et que les résultats sont cohérents, vous pourrez vous interroger sur la qualité de vos emploi du temps.

Par exemple les étudiants n'aiment généralement pas avoir de "trous" dans leur emploi du temps. On peut considérer que pour un parcours, un cours correspond un un créneau dans une journée durant lequel il n'y a aucun cours, alors qu'il existe le même jour au moins deux autres créneaux avant et après celui considéré, durant lequel un cours est programmé pour ce parcours.

3. Proposer un modèle `edt_minTrous.mod` permettant de minimiser le nombre total de trous pour l'ensemble des parcours de l'emploi du temps.
4. Essayer de réfléchir à deux autres critères distincts pouvant être pris en compte pour essayer d'améliorer encore la qualité des emploi du temps. Justifier votre proposition expliquez comment les modéliser et proposer un modèle `edt_minTrous2.mod` permettant permettant de combiner ces différents critères en précisant la façon de les combiner.

4 Organisation du code et caractérisation des résultats

Vous trouverez sur la page du cours sur eCampus, une archive, que vous pourrez rajouter dans le répertoire src/opl de votre projet, possédant la structure suivante :

```
devoirEdt/
     ...  les fichiers de code opl (fichiers mods)
     data/
          instances/
          tests/
               instance_type.dat
    js/
    rapport/
    resultats/
          instances/
          tests/
```

Vous pourrez partir de cette archive pour la compléter mais il vous est demandé de ne pas en modifier la structure.

- Tous vos fichiers `.mod` devront se trouver au niveau du repertoire `devoirEdt`. Notez que vous pouvez découper votre code en plusieurs fichiers de façon à mutualiser des parties communes du code (en utilisant des `include`).
- le répertoire `js` est destiné à recevoir d'éventuels fichiers en langage de script, que vous pourriez avoir envie du mutualiser entre différents modèles. Ils peuvent par exemple contenir n'importe quel séquence de langage de script, notamment des définitions de fonctions. Un fichier en langage de script peut être inclus dans un bloc de script `execute{...}` en utilisant la fonction `includeScript(path_to_file)`. Si vous n'en ressentez pas le besoin, vous pouvez laisser ce répertoire vide.
- Le répertoire `data`, contiendra uniquement des fichiers .dat décrivant des données décrivant des instances. Vous pouvez structurer son contenu comme vous le souhaitez mais l'idée est de séparer les fichiers servant juste à faire des tests lors de la mise au point de vos modèles (que vous mettrez dans le sous-repertoire `tests/`) de ceux qui pourraient constituer des exemples complets de problèmes (à placer dans le sous-repertoire `tests/`). vous pouvez rajouter d'autres sous-répertoires pour mieux structure les instances par catégories.
- Comme sont nom l'indique, Le répertoire `resultats`,, contiendra uniquement des fichiers correspondant aux résultats obtenus à l'aide de différents modèles sur des fichiers d'instances.
  Afin de pouvoir vérifier (automatiquement) que les emplois du temps générés par vos modèles sont corrects, on vous demande de sauvegarder les résultats produits par vos modèles en utilisant la syntaxe OPL, sous la forme d'un fichier `.dat` dans le répertoire `resultats`.
  L'idée est de garder dans ced répertoire la même organisation structurelle que celle du répertoire `data`. On prendra comme convention que si un fichier d'instance dont le chemin relatif (qui correspond aussi au nom de l'instance) est de la forme `data/path/nomFichier.dat` alors le résultat obtenu en utilisant un modèle `edt_XXX.mod` sera sauvegardé dans le fichier `resultat/path/nomFichier_edt_XXX.dat`, suivant le format décrit ci-dessous.
  Exemple : si l'on applique le modèle `edt_salles.mod` sur le fichier d'instance `data/test/instance_type.dat`, le résultat correspondant doit être sauvegardé dans le fichier `resultat/test/instance_type_edt_salles.dat`
  Notez que pour pouvoir respecter cette consigne, il est impératif que le nom donné à chaque instance soit bien une chaine de caractère correspondant au chemin relatif depuis le repertoire de base, jusqu'au fichier décrivant l'instance. Vous serez amenés à décomposer cette chaine de caractère pour reconstruire celle correspondant au chemin du fichier résultat.
- Le répertoire `rapport` est destiné à recevoir un petit rapport au format pdf (exclusivement) d'une dizaine de page au maximum et faisant la synthèse de votre travail et dont la qualité sera prise en compte dans la note du devoir.

Format de sauvegarde des résultats

Pour sauvegarder les données décrivant l'emploi du temps produit par un solveur, on vous demande d'écrire dans le fichier .dat correspondant au minimum deux informations :

- nomInstance : une chaine de caractère contenant le nom (i.e. le chemin d'acces) de l'instance traitée
- edt : un ensemble de tuples de type `edtCours` décrivant les caractéristiques de chaque cours et correspondant à la structure présentée ci-dessous.

```
structure EdtCours {
   string idCours ;        // identifiant du cours
   int  jour ;             // jour du cours
   int   creneauJour ;     // créneau du cours pour ce jour (de 1 à nbCrenaux)
   int    h ;              // heures de l'horaire de début du cours
   int    mn ;             // minutes de l'horaire de début du cours
   string idSalle;         //  salle du cours
   }
```

Pour le modèle `edt_simple.mod`, comme les salles ne sont pas affectées, vous pourrez simplement mettre une chaine de caractère vide, pour la dernière compostante.

Vous pourrez rajouter toues les informations jugées utiles sous la forme d'un plusieurs commentaires, délimités par `/* ... */` comme par exemple une présentation plus lisible de la solution, par ordre chronologique ou encore, pour les problèmes d'optimisation, la valeur optimale du critère d'optimisation.

A Exemple type de fichier d'instance

```
/* Exemple illustrant la sytaxe des fichiers d'instances */
 nomInstance = "data/tests/instance_type.dat"; // Chemin relatif vers ce fichier
nbJours = 15;                                  // Nombre de jours de l'emploi du temps
nbCreneauxParJour = 8;                         // Nombre de créneaux par jour
dureeCreneau = 60;                             // Durée d'un créneau (en mn)
horairePremierCreneau = <8,30>;                // Horaire du premier créneau d'une journée

filieresEdt = {
/*  idFiliere            coursCommuns                          parcours
/* ------------------------------------------------------------------------------------ */
   <"L3-Informatique",   {"PFA","LogiqueL3"},       {"L3INP1", "L3INP2" }>,
   <"L3-Miage",  {"LogiqueL3","EcoL3IG","IG-Conf1"}, {"L3IGP1", "L3IGP2", "L3IGP3" }>
};

parcoursEdt = {
/*       idParcours  Effectifs   coursParcours
/* ------------------------------------------------------------------------------------ */
   < "L3INP1",  24,          {"PFA-TP1", "LogL3-TD1"} > ,
   < "L3INP2",  26,          {"PFA-TP2", "LogL3-TD2"} > ,
   //
   <"L3IGP1",   20,          {"EcoL3IG-TD1", "LogL3-TDIG1"} >,
   <"L3IGP2",   10,          {"EcoL3IG-TD2", "LogL3-TD1"}  >,
   <"L3IGP3",   8,           {"EcoL3IG-TD2", "LogL3-TD2"}  >
};


coursEdt  = {
/*      Code          Description    Prof      Duree(mn)   Besoins */
/* ------------------------------------------------------------------------------------ */
 <"PFA",        "Programmation Fonctionnelle Avancée - Cours"
                              "TomP",    90,        {"video", "amphi"}>,
 <"PFA-TP1",    "Programmation Fonctionnelle Avancée - TP1",
                              "TomP",   120,        {"tp_info"}>,
 <"PFA-TP2",    "Programmation Fonctionnelle Avancée - TP21",
                              "JeanF",  120,        {"tp_info"}>,
 <"LogiqueL3",  "Logique pour l'informatique  - Cours",
                              "ChrisP",  90,        {"video", "amphi"}>,
 <"LogL3-TD1",  "Logique pour l'informatique - TD 1",
                              "AlexT",   120,       {"td"}>,
 <"LogL3-TD2",  "Logique pour l'informatique - TD 2",
                              "AlexT",   120,       {"td"}>,
 <"LogL3-TDIG1","Logique pour l'informatique - TD 3",
                              "FranckS", 120,       {"td"}>,
 <"EcoL3IG",    "Economie - Cours","GregD", 120,     {"video","cours","td"}>,
 <"EcoL3IG-TD1","Economie - TD1",  "GregD", 90,      {"td","cours"}>,
 <"EcoL3IG-TD2","Economie - TD2",  "GregD", 90,      {"td","cours"}>,
 <"IG-Conf1",   "Conference Droit de l'informatique",
                              "Conf1",   150,       {"cours"}>,
};

seriesCours  = {
/*      idCours   nbOccurences  ecartMin  ecartMax */
/* ------------------------------------------------------------------------------------ */
 <"PFA",           3,      5,      5>,
 <"PFA-TP1",       3,      4,      6>,
```

```
 <"PFA-TP2",          3,       4,       6>,
 <"LogiqueL3",        3,       5,       5>,
 <"LogL3-TD1",        3,       5,       5>,
 <"LogL3-TD2",        3,       5,       5>,
 <"LogL3-TDIG1",      3,       5,       5>,
 <"EcoL3IG",          3,       2,       3>,
 <"EcoL3IG-TD1",      3,       2,       3>,
 <"EcoL3IG-TD2",      3,       2,       3>,
};


salles = {
/*  idSalle        capacite  services */
/* ------------------------------------------------------------------------------ */
  <"PUIO-GrandAmphi", 250, {"video", "amphi", "cours"}>,
  <"PUIO-PetitAmphi", 100, {"video", "amphi", "cours"}>,

  <"PUIO-C203",        40, {"video", "td"}>,
  <"PUIO-C105",        40, {"td"}>,
  <"PUIO-C204",        50, {"video", "cours", "td"}>,
  <"PUIO-C202",        24, {"video", "tp_info"}>,
  <"PUIO-C101",        26, {"video", "tp_info"}>,
};

indisponibilitesProfs = {
/*  idProf    jour      horaireDebut    horaireFin */
/* ------------------------------------------------------------------------------ */
   <"TomP",     1,    <8,30>,         <16,30> >,
   <"JeanF",    4,    <7,00>,         <10,00> >,
   <"JeanF",   12,    <13,00>,        <18,30> >,
};

indisponibilitesSalles = {
*  idSalle     jour      horaireDebut    horaireFin */
/* ------------------------------------------------------------------------------ */
        <"PUIO-GrandAmphi", 1,   <8,30>, <18,30> >
        <"PUIO-C204",   2,   <14,30>, <18,30> >
};

precedences = {
        /* idCoursAvant, idCoursApres */
        <"PFA",                  "PFA-TP1">,
        <"PFA",                  "PFA-TP2">,
        <"LogiqueL3",    "LogL3-TD1">,
        <"LogiqueL3",    "LogL3-TD2">,
        <"LogiqueL3",    "LogL3-TDIG1">
};

memesHoraires = {
   /* idCours1,     idCours2 */
   <"PFA-TP1",      "PFA-TP2">
};

creneauxFixes = {
        /* idCours  jour,  seance creneau */
        <"IG-Conf1", 2,      2,     3>
};
```