Mini-Projet d'Intelligence Artificiel
Emploi du temps à Polytech Nantes
-
Rapport complet

Guillaume Clochard, Thomas Coquereau

14 novembre 2018

1 Introduction

Ce rapport présente un premier travail effectué dans le cadre du Mini-Projet d'Intelligence Artificielle. Il consiste en la planification de l'emploi du temps à Polytech Nantes. Ce présent document contient le premier rendu (Résolution générale du problème), ainsi que la suite du projet (Résolution du problème en Prolog).

Il est également fournit le code source du projet.

```
README.md
instance.pl
instance_old.pl
main.pl
planifier.pl
run_test.pl
tests\
    instance.pl
    planifier.pl
    utils.pl
utils.pl
```

Listing 1 – Fichiers

Le programme se lance par la commande suivante :

```
swipl -s main.pl
```

Listing 2 – Lancer l'algorithme

Les tests unitaires :

```
swipl -s run_tests.pl
```

Listing 3 – Lancer les tests

2 Résolution générale du problème

2.1 UML

Voici dans la Figure **??** le diagramme de classe décrivant les données nécessaires à la gestion de l'emploi du temps.

Comme on peut le constater, on regroupe l'ensemble des données nécessaires et déterminées à l'avance dans des Séances. A savoir : le(s) groupe(s) d'étudiants concerné(s), le(s) enseignant(s), la matière, ainsi que le type de cours. Ces séances vont ensuite être associées à des créneaux par notre solution. Les créneaux étant le regroupement d'un jour, une plage horaire et une salle.

Un peu noter des détails intéressants, sur Groupe, il y a la notion d'incompatibilité qui permet de définir lorsqu'il est possible pour deux groupes d'avoir cours sur une même plage horaire. Sur matière on a la notion de suite, lorsqu'une matière débute après la fin d'une autre (le Mini-Projet d'IA qui suit par exemple le cours d'IA). Et enfin sur séance, on a la notion de suite aussi qui décrit une nombre de jours minimum et maximum avant le prochain cours de la même matière.

Remarque. Les noms données aux objets de la modélisation sont légèrement différents de la solution donnée lors de l'indication de mi-parcours. Nous appelons par exemple "Créneau" l'objet à construire et regroupant Séance, Salle, Moment.

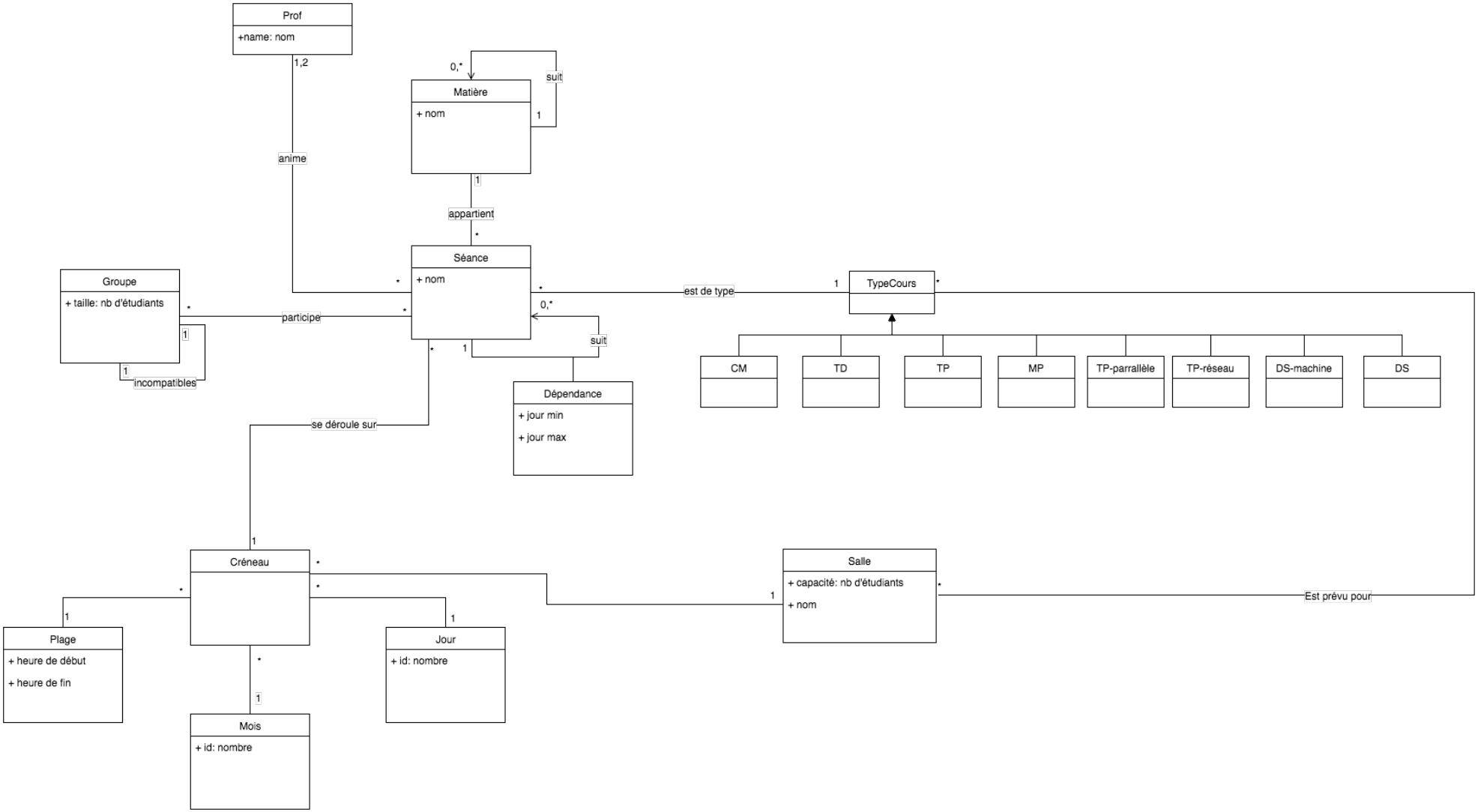


FIGURE 1 – Modélisation UML

2.2 Instance

Le problème modélisé, il convient maintenant de construire un jeu de données qui servira à la construction de l'emploi du temps.

Deux instances sont disponibles :

`instance_old.pl` Un petit jeu de données (17 séances) nous ayant servit au début du projet et servant actuellement pour les tests unitaires

`instance.pl` Représente un semestre complet (458 séances, 21 matières, 32 enseignants, 9 groupes, 20 salles)

Pour générer `instance.pl`, des prédicats ont été utilisés afin de faciliter la saisie des données en regroupant les séances d'une même matière dans une même déclaration.

2.3 Solution

Il convient à présent de définir une première proposition de solution.

Imaginons qu'on dispose d'une fonction Planifier en charge de créer l'emploi du temps.

Sa signature serait grossièrement :

Entrées les séances S, les plages horaires H, les jours de travail J, les mois M et les salles L

Sortie un sous-ensemble de tous les créneaux C

$$\text{Planifier} : S \times H \times J \times L \quad \not\to \quad C$$

On retrouve donc les objets modélisés à la Figure **??**.

2.3.1 Pré-conditions

Les pré-conditions de la fonction Planifier sont formées par le respect de la modélisations UML et de l'ajout de détails comme :

— les jours sont des nombres entiers appartenant à $\mathbb{N}_+$
— les plages horaires ne débordent pas des 24h et ne se recouvrent pas

$$\forall (h_0, h_1) \in H, h_0 \leq 0h00, h_1 \leq 23h59$$

$$\forall (h_0, h_1), (h_2, h_3) \in H,\ h_0 < h_1 \leq h_2 < h_3 \text{ ou } h_2 < h_3 \leq h_0 < h_1$$

— tous les types de cours des séances voient une salle compatible

$$\forall s \in S, \exists l \in L, \text{typeCours}(s) = \text{typeCours}(l)$$

2.3.2 Post-conditions

— les créneaux ne débordent pas sur des horaires, des jours ou des salles différents des entrés

$$\forall (s, h, j, l) \in \text{Planifier}(S, H, L, L), s \in S, h \in H, j \in J, l \in L \tag{P1}$$

— toutes les séances sont affectées à un créneau

$$\forall s \in S, \exists (s, h, j, l) \in \text{Planifier}(S, H, L, L) \tag{P2}$$

— toute séance est affectée à une salle qui respecte son type de cours et son effectif

$$\forall (s, h, j, l) \in \text{Planifier}(S, H, L, L),\ \text{typeCours}(s) = \text{typeCours}(l) \text{ et effectif}(s) \leq \text{taille}(l) \tag{P3}$$

— un enseignant n'a pas deux séances en même temps

$$\forall (s_1, h, j, l_1), (s_2, h, j, l_2) \in \text{Planifier}(S, H, L, L),\ \text{prof}(s_1) \neq \text{prof}(s_2) \tag{P4}$$

— des groupes incompatibles n'ont pas cours en même temps

$$\begin{aligned}&\forall(s_1,h,j,l_1),(s_2,h,j,l_2)\in \textsc{Planifier}(S,H,L,L),\\&\neg\text{incompatibles}(g_1,g_2)\quad \forall g_1\in groupes(s_1),\forall g_2\in groupes(s_2)\end{aligned}\tag{P5}$$

— les créneaux respectent le séquencement des séances et des matières

$$\begin{aligned}&\forall(s_1,h_1,j_1,l_1),(s_2,h_2,j_2,l_2)\in \textsc{Planifier}(S,H,L,L),\text{tel que suit}(s_2,s_1)\\&\text{alors } j_2\in O \text{ où } O=intervalleJours(s_1,s_2)\\&\text{et si } j_2=j_1 \text{ alors } h_2>h_1\end{aligned}\tag{P6}$$

2.3.3 Algorithme non déterministe

Nous avons choisit de formuler notre solution sous forme d'un algorithme non déterministe.

Dans l'algorithme **??**, on construit des créneaux liant correctement une séance, un horaire, un jour ainsi qu'une salle à l'aide de l'*oracle* `choix_nd`.

À chaque nouveau choix, on teste si les post-conditions P_i données au-dessus sont respectées. Si c'est le cas, on ajoute le créneau composé de ces données dans la liste des créneaux et on enlève la séance de la liste des séance. Si ces les post-conditions ne sont pas respectées par le nouvel ajout, on retourne $\perp$ afin de marquer l'échec de ce créneau.

On continue ainsi jusqu'à ce qu'il n'y ait plus de séances à placer.

$C \Leftarrow \emptyset$
while $|S| > 0$ **do**
  $s \Leftarrow choix\_nd(S)$
  $h \Leftarrow choix\_nd(H)$
  $j \Leftarrow choix\_nd(J)$
  $l \Leftarrow choix\_nd(L)$
  $c \Leftarrow (s,h,j,l)$
  if les post-conditions P_i sont respectées avec c **then**
    $C \Leftarrow c \cup C$
    $S \Leftarrow S \backslash s$
  else
    return $\perp$
  end if
end while

Algorithm 1: Proposition de solution

3 Résolution du problème en Prolog

3.1 Algorithme

3.1.1 Résolution

Voici dans le code **??** ci-dessous l'algorithme principal `planifier(+Ss, +Ds, -Cs)`. L'idée est de découper le problème en sous-problèmes par récursion. La planification d'une séance demande ainsi la planification des séances la précédant dans la liste `Ss` initiale (sous-problème).

```
/**
 * planifier(+Ss, +Ds, -Cs).
 *
 * @arg Ss  Listes des seances a planifier
 * @arg Ds  Listes des creneaux deja planifie
 * @arg Cs  Listes des creneaux construits
 */
planifier([], _, []) :- !.
planifier(Ss, Ds, [C|Cs]) :-

    member(S, Ss),       % La seance courante
    delete(Ss, S, Ss2), % On l'enleve de la liste

    planifier(Ss2, Cs), % on traite le sous-probleme

    append(Ds, Cs, Cs2), % on cree une liste qui regroupe les creneaux existants

    % Creation du creneau et tests ----------------------------------------

    seance(S, TypeS, _, _),

    date(J, M),      % une date
    plage(H, _, _), % une plage horaire

    % une salle
    salle(L, TailleL),
    accueille(L, TypeL),
    typesCoursIdentiques(TypeS, TypeL), % type de salle valide

    findall(G, groupeSeance(G, S), Gs), % tous les groupes de la seance
    effectifGroupes(Gs, Effectif),
    Effectif =< TailleL, % taille de salle valide

    findall(P, profSeance(P, S), Ps),   % tous les enseignants de la seance

    % test des contraintes (profs, incompatibilite groupes, sequencement)
    % sur cette proposition de creneau
    creneauValide(S, Ps, Gs, H, J, M, L, Cs2),

    % Fin creation du creneau et tests ------------------------------------

    C = [S, H, J, M, L].
```

Listing 4 – Resolution

La planification d'une séance consiste donc à faire les choix non déterministes présentés dans l'algorithme **??**. L'ordre d'éxécution de ces choix a son importance : le back-tracking de Prolog remontera sur les choix pour trouver une combinaison validant les conditions dans l'ordre que nous avons défini.

Pour chaque séance, on détermine donc tout d'abord le moment, puis la salle, avant de tester si le créneau formé est valide. Si ce n'est pas le cas, Prolog testera toutes les autres salles avant de rechercher une autre date. Choisir l'ordre inverse (choix de la salle avant la date) donnerait un emploi du temps s'étalant sur une période

plus longue, Prolog préférant alors passer à un moment suivant plutôt que de cherche une salle libre (ie. peu de cours en parallèle).

Exemple : le 01/01, de 14h à 15h30, on peut avoir plusieurs séances avec des groupes différents et des salles différentes.

À chaque propositon de créneau, on effectue les tests de validité des posts-conditions :

- On vérifie que la salle le bon type de cours.
- On se penche ensuite sur les problèmes de groupes. On récupère les groupes associés à la séance, et on vérifie que la salle convient à l'effectif .
- Les autres post-conditions nécessitent de parcourir les créneaux déjà construits. C'est `creneauValide` qui s'en charge. On l'explique plus bas.

Il est donc important d'effectuer les tests au plus tôt après le choix non déterministe. On optimise ainsi les calculs nécessaire. Prolog ne lancera par exemple pas le `findall` des groupes tant qu'une salle de bon type n'est pas trouvée.

3.1.2 Prédicats utilitaires

Des prédicats déterministes ont été implémentés afin de participer à la validation des post-conditions et à l'ajout d'un créneau potentiel.

```
/**
 * creneauValide(+S, +Ps, +Gs, +H, +J, +M, +L, +Cs).
 *
 * Definit si un creneau est valide (Pas de conflit avec les creneaux existants)
 *
 * @arg S   La seance
 * @arg Ps  Les enseignants
 * @arg Gs  Les groupes
 * @arg H   La plage horaire
 * @arg J   Le jour
 * @arg M   Le mois
 * @arg L   La salle
 * @arg Cs  Liste de creneaux [S, H, J, M, L]
 */
creneauValide(_, _, _, _, _, _, _, []) :- !.
creneauValide(S, Ps, Gs, H, J, M, L, [C|Cs]) :-
    creneauValideCreneau(S, Ps, Gs, H, J, M, L, C),
    creneauValide(S, Ps, Gs, H, J, M, L, Cs),
    !.
```

Listing 5 – creneauValide

Le code du listing **??** montre la logique récursive de la validation d'un nouveau créneau.

```
/**
 * creneauValideCreneau(+S, +Ps, +Gs, +H, +J, +M, +L, +C).
 *
 * Definit si un cours n'est pas incompatible au moment donne avec les autres
 * cours de la liste de creneaux.
 *
 * @arg Ps  Les enseignants
 * @arg Gs  Les groupes
 * @arg H   La plage horaire
 * @arg J   Le jour
 * @arg M   Le mois
 * @arg L   La salle
 * @arg C   Un creneau [S, H, J, M, L]
 */
creneauValideCreneau(S, Ps, Gs, H, J, M, L, C) :-
    % le creneau valide le sequencement avec C
    sequencementValideCreneau(S, H, J, M, C),
```

```
    (
        % le creneau n'est pas au meme moment que C
        (\+ memeMomentCreneau(H, J, M, C));
        % ou il ne concerne pas un meme prof, des groupes incompatibles
        % ou une meme salle
        (
            \+ groupesIncompatibleCreneau(Gs, C),
            \+ memeProfs(Ps, C),
            \+ memeSalle(L, C)
        )
    ),
    !.
```

Listing 6 – creneauValideCreneau

Le code du listing **??** montre les tests effectués sur chaque créneau. Les autres prédicats déterministes sont documentés dans `planifier.pl` :

- `sequencementValideCreneau(+S, +H, +J, +M, +C)` vérifie que le nouveau créneau respecte les contraintes de séquencement avec le créneau `C` existant.
- `\+ memeMomentCreneau(+H, +J, +M, +C)` teste si les deux créneaux ont lieu en même temps.
- `\+ groupesIncompatibleCreneau(+Gs, +C)` vérifie que les groupes ne sont pas incompatibles avec les groupes du créneau C.
- `\+ memeProfs(+Ps, +C)` vérifie que tous les profs du créneau sont disponibles.
- `\+ memeSalle(+L, +C)` vérifie que la salle est libre.

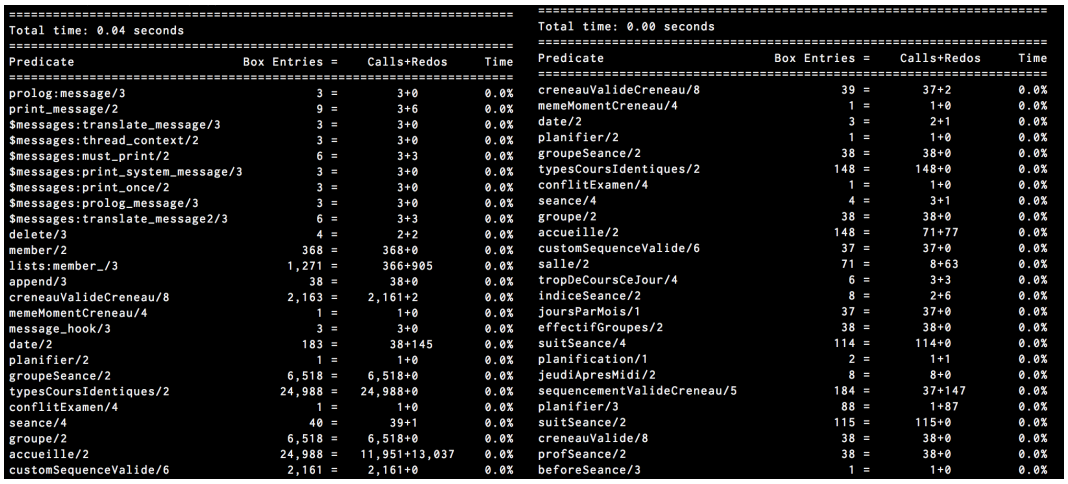
3.2 Ordonnancement et heuristique

Il est important de remarquer que les séances peuvent être ordonnées avant de chercher une planification. Les séances d'une même matière se suivant, il est intéressant de commencer par planifier la première, puis la seconde, etc.

Le prédicat `indiceSeance(+S, -I)` implémente donc l'heuristique retournant l'indice d'une séance dans l'arbre des séances séquencées. Cette heuristique nous permet de trier les séances à priori et de commencer la la planification des séances de plus petits indices. Elles doivent logiquement se dérouler en début de période scolaire.

Cet ordonnancement nous permet de rapidement partir dans une branche relativement bonne. Sans ordonnancement, on peut voir le moteur d'inférence commencer par placer la dernière séance d'une matière sur le premier créneau de l'année, ce qui entraine alors de très nombreux retours arrières.

Cela revient un peu à trier les séances selon un critère de contrainte (leur dépendance aux séances précédentes). L'originalité est ici qu'on ne commence pas par les plus contraintes, mais par les plus libres, à cause de l'aspect temporel du problème.



Total time: 0.04 seconds

Predicate	Box Entries	Calls+Redos	Time
prolog:message/3	3	3+0	0.0%
print_message/2	9	3+6	0.0%
$messages:translate_message/3	3	3+0	0.0%
$messages:thread_context/2	3	3+0	0.0%
$messages:must_print/2	6	3+3	0.0%
$messages:print_system_message/3	3	3+0	0.0%
$messages:print_once/2	3	3+0	0.0%
$messages:prolog_message/3	3	3+0	0.0%
$messages:translate_message2/3	6	3+3	0.0%
delete/3	4	2+2	0.0%
member/2	368	368+0	0.0%
lists:member_/3	1,271	366+905	0.0%
append/3	38	38+0	0.0%
creneauValideCreneau/8	2,163	2,161+2	0.0%
memeMomentCreneau/4	1	1+0	0.0%
message_hook/3	3	3+0	0.0%
date/2	183	38+145	0.0%
planifier/2	1	1+0	0.0%
groupeSeance/2	6,518	6,518+0	0.0%
typesCoursIdentiques/2	24,988	24,988+0	0.0%
conflitExamen/4	1	1+0	0.0%
seance/4	40	39+1	0.0%
groupe/2	6,518	6,518+0	0.0%
accueille/2	24,988	11,951+13,037	0.0%
customSequenceValide/6	2,161	2,161+0	0.0%

(a) Sans ordonnancement

Total time: 0.00 seconds

Predicate	Box Entries	Calls+Redos	Time
creneauValideCreneau/8	39	37+2	0.0%
memeMomentCreneau/4	1	1+0	0.0%
date/2	3	2+1	0.0%
planifier/2	1	1+0	0.0%
groupeSeance/2	38	38+0	0.0%
typesCoursIdentiques/2	148	148+0	0.0%
conflitExamen/4	1	1+0	0.0%
seance/4	4	3+1	0.0%
groupe/2	38	38+0	0.0%
accueille/2	148	71+77	0.0%
customSequenceValide/6	37	37+0	0.0%
salle/2	71	8+63	0.0%
tropDeCoursCeJour/4	6	3+3	0.0%
indiceSeance/2	8	2+6	0.0%
joursParMois/1	37	37+0	0.0%
effectifGroupes/2	38	38+0	0.0%
suitSeance/4	114	114+0	0.0%
planification/1	2	1+1	0.0%
jeudiApresMidi/2	8	8+0	0.0%
sequencementValideCreneau/5	184	37+147	0.0%
planifier/3	88	1+87	0.0%
suitSeance/2	115	115+0	0.0%
creneauValide/8	38	38+0	0.0%
profSeance/2	38	38+0	0.0%
beforeSeance/3	1	1+0	0.0%

(b) Avec ordonnancement

FIGURE 2 – Comparaison sans/avec ordonnancement

Sur la figure **??**, on compare deux profiling d'un jeu de données très simple (2 séances S1 et S2, S2 suit S1), à gauche, sans ordonnancement, à droite, avec. On remarque par exemple un nombre bien plus important de "Calls+Redos" sur `creneauValideCreneau` ou `typesCoursIdentiques` à gauche.

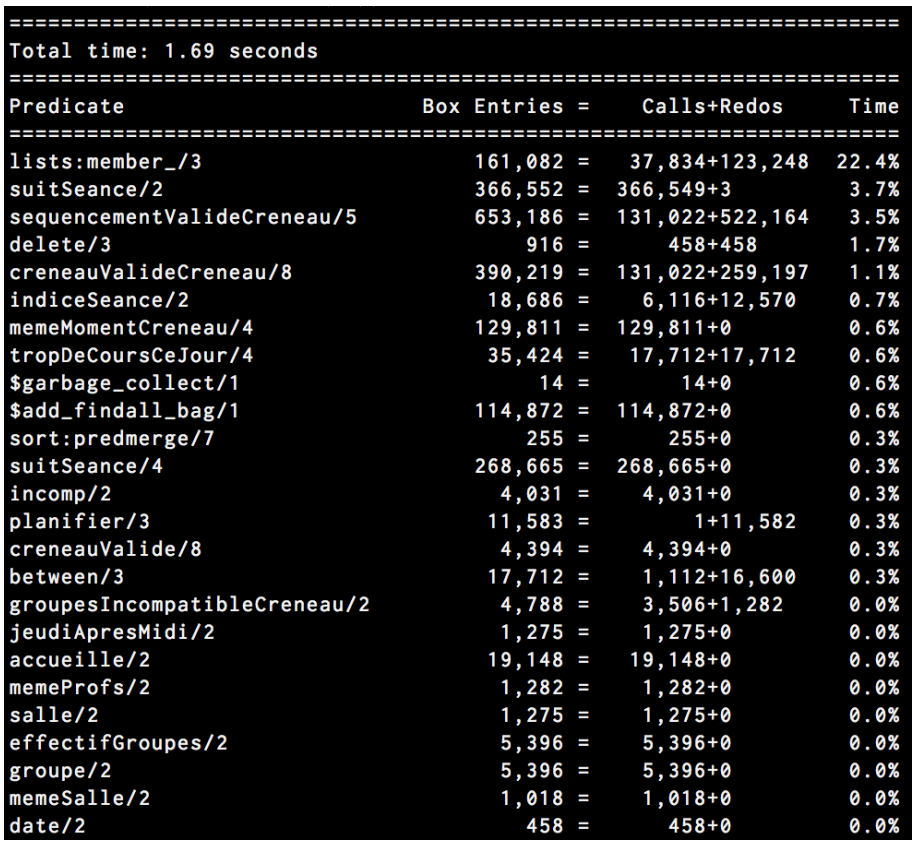
En effet, l'algorithme a eu le malheur d'assigner la séance S2 au premier moment disponible et à tester toutes les créneaux possibles pour S1 avant d'effectuer un retour arrière et mettre S1 en premier. C'est le genre de situation que l'ordonnancement permet d'éviter.

3.3 Performances

Les performances de l'algorithme final sur notre jeu de données complet (458 séances, 21 matières, 32 enseignants, 9 groupes, 20 salles) peuvent être capturées avec `profile(planification(Cs))`.

Le résultat est à la figure **??**.

On remarque donc un temps de 1,68 secondes sur une machine donnée. Les performances peuvent être améliorées encore, on le voit, en réduisant l'usage du prédicat `member/2`. C'est par exemple notre prédicat `accueille(+Salle, ?TypeCours)`, très utilisé, qui en fait un usage important. Cette donnée pourrait être déclaré au lancement du programme par `assert/1`.



```
================================================================================
Total time: 1.69 seconds
================================================================================
Predicate                         Box Entries =    Calls+Redos     Time
================================================================================
lists:member_/3                       161,082 =   37,834+123,248  22.4%
suitSeance/2                          366,552 =  366,549+3         3.7%
sequencementValideCreneau/5           653,186 =  131,022+522,164   3.5%
delete/3                                  916 =      458+458       1.7%
creneauValideCreneau/8                390,219 =  131,022+259,197   1.1%
indiceSeance/2                         18,686 =    6,116+12,570    0.7%
memeMomentCreneau/4                   129,811 =  129,811+0         0.6%
tropDeCoursCeJour/4                    35,424 =   17,712+17,712    0.6%
$garbage_collect/1                         14 =       14+0         0.6%
$add_findall_bag/1                    114,872 =  114,872+0         0.6%
sort:predmerge/7                          255 =      255+0         0.3%
suitSeance/4                          268,665 =  268,665+0         0.3%
incomp/2                                4,031 =    4,031+0         0.3%
planifier/3                            11,583 =        1+11,582    0.3%
creneauValide/8                         4,394 =    4,394+0         0.3%
between/3                              17,712 =    1,112+16,600    0.3%
groupesIncompatibleCreneau/2            4,788 =    3,506+1,282     0.0%
jeudiApresMidi/2                        1,275 =    1,275+0         0.0%
accueille/2                            19,148 =   19,148+0         0.0%
memeProfs/2                             1,282 =    1,282+0         0.0%
salle/2                                 1,275 =    1,275+0         0.0%
effectifGroupes/2                       5,396 =    5,396+0         0.0%
groupe/2                                5,396 =    5,396+0         0.0%
memeSalle/2                             1,018 =    1,018+0         0.0%
date/2                                    458 =      458+0         0.0%
```

FIGURE 3 – Résultat du profiling

3.4 *One more thing*

3.4.1 `dynamic`

Afin de saisir relativement rapidement les séances correspondantes à un semestre complet, nous avons utilisé des prédicats travaillant sur des listes. L'inconvénient est la charge de travail supplémentaire demandé à Prolog à chaque prédicat `seance/4`, par exemple. Toutes ces données étant disponibles à priori, nous avons donc utilisé les prédicats Prolog `dynamic` et `assertz/1` qui permettent de définir des prédicats à la volée, comme un chargement de base de données.

Cette transformation a permit un gain d'un ordre de grandeur 100, passant de 500 secondes (9min) à 2s pour la planification du même jeu de données. On peut comparer l'ancien profiling (Figure **??**) avec le nouveau (Figure **??**). À remarquer : le temps important passé dans les prédicats gravitant autour du concept de séance sans utiliser `dynamic`.

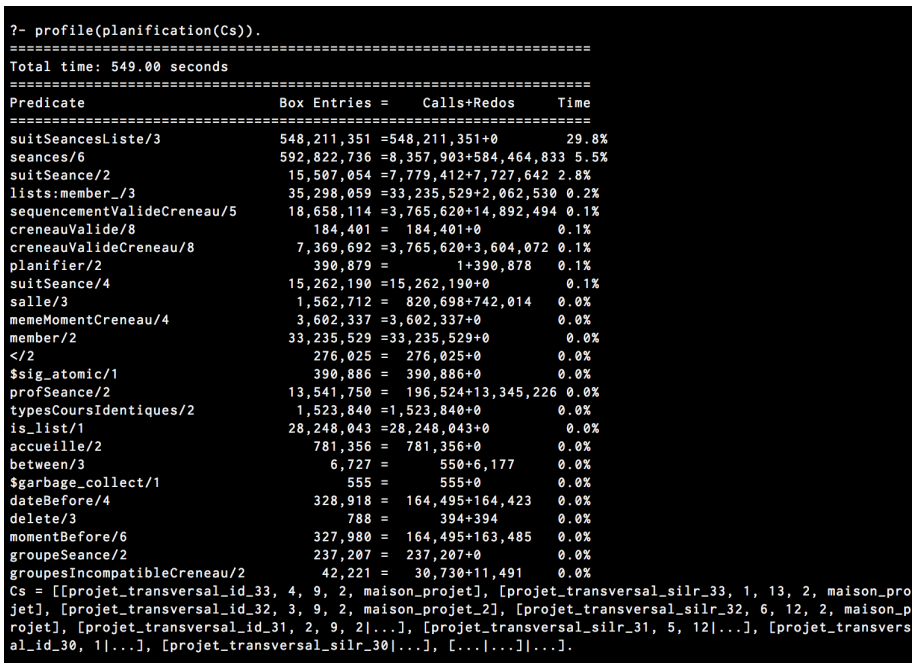


FIGURE 4 – Résultat du profiling avant dynamic

3.4.2 Optimisation de l'emploi du temps

Afin de rendre l'emploi du temps un peu plus vivable, on peut optimiser un peu les choses.

Évitons de planifier des cours en début ou fin de journée :

```
% plage(H, _, _).
member(H, [2, 3, 4, 5, 1, 6]), % une plage horaire
                               % evitons premier et dernier creneau
```

Listing 7 – Éviter début et fin de journée

Limitons le nombre de cours par jours :

```
/**
 * tropDeCoursCeJour(+J, +M, +Max, +Cs)
 *
 * @arg J   Jour
 * @arg M   Mois
 * @arg Max Nombre max de cours par jour
 * @arg Cs  Liste de creneaux deja planifies
 */
tropDeCoursCeJour(J, M, Max, Cs) :-
    findall(S, member([S, _, J, M, _], Cs), Csjour),
    length(Csjour, X),
    X > Max.
```

Listing 8 – Limite de cours par jour

Empêchons les cours le jeudi après-midi :

```
/**
 * jeudiApresMidi(+H, +J)
 *
 * Est vrai si le moment passe est un jeudi apres midi
 *
 * Partant du principe qu'il y a 5 jours dans une semaine et que l'annee
 * commence un lundi, sans cas particulier, alors les jeudis tombent le 4eme
 * jour de chaque semaine.
 *
```

```
 * @arg H    Plage horaire
 * @arg J    Jour
 */
jeudiApresMidi(H, J) :-
    (J + 1) mod 5 is 0,
    H > 3.
```

Listing 9 – Jeudi après-midi

Limitons à un examen par jour :

```
/**
 * conflitExamen(+S, +J, +M, +C)
 *
 * Est vrai si C se deroule le meme jour et que les deux creneaux sont des
 * examents
 *
 * @arg S    Seance
 * @arg J    Jour
 * @arg M    Mois
 * @arg C    Un creneau
 */
conflitExamen(S, J, M, [S2, _, J, M, _]) :-
    seance(S, ds, _, _),
    seance(S2, ds, _, _).
```

Listing 10 – Détection de conflit d'exament

L'intégration de ces prédicats est disponible dans le code joint au rendu.

3.4.3 Affichage

```
/**
 * afficherPlanification(+Cs)
 *
 * Affiche la planification dans l'ordre chronologique
 *
 * @arg Cs  Une liste de creneaux
 */
afficherPlanification([]) :- !.
afficherPlanification(Cs) :-
    date(J, M),
    plage(H, _, _),
    member(C, Cs),
    C = [S, H, J, M, L],
    write('------------------------------------------------------------'), nl,
    afficherSeance(S), nl,
    afficherMoment(J, M, H), nl,
    afficherGroupes(S), nl,
    afficherProfs(S), nl,
    afficherSalle(L), nl,
    delete(Cs, C, Cs2),
    afficherPlanification(Cs2),
    !.
```

Listing 11 – Affichage

Pour l'affichage, on utilise la fonction `afficherPlanification(+Cs)` que l'on peut voir dans l'algorithme **??** ci-dessus.

On va chercher à afficher l'emploi du temps par ordre chronologique : donc du premier jour, première plage jusqu'au dernier jour, dernière plage.

De ce fait, la fonction récupère tout d'abord la date puis, dans un second temps, une plage.

Ensuite, elle sélectionne un des créneaux qui correspond et débute l'affichage.

```
/**
 * afficherSeance(+S)
 *
 * @arg S
 */
afficherSeance(S) :-
    seance(S, TypeCours, Mat, Nom),
    write('Seance:\t\t'),
    write(S), write('␣"'), write(Nom), write('"'),
    write('␣–␣'), write(TypeCours),
    write('␣–␣'), write(Mat).
```

Listing 12 – Affichage

On affiche tout d'abord la séance (nom, type, matière), puis le moment (jour, mois et plage) et ainsi de suite jusqu'à avoir toutes les informations propres à un emploi du temps pour ce créneau. Sur l'algorithme **??** ci-dessus, on a un exemple de fonction : celle permettant d'afficher les séances.

On supprime ensuite ce créneau de la liste des créneaux à traiter puis on rappelle récursivement la fonction sur la nouvelle liste.

On peut voir le rendu de l'affichage sur l'image **??**.

```
---------------------------------------------------------------
Séance:         projet_transversal_silr_17 "PROJET projet transversal" - projet - projet_transversal
Date:           9/4 09h45-11h15
Groupes:        silr
Profs:          prof_ptrans_silr
Salle:          maison_projet(1000)
---------------------------------------------------------------
Séance:         tp_c_id_8 "TP C++" - tp - logiciel_c
Date:           9/4 09h45-11h15
Groupes:        id
Profs:          soufiane
Salle:          b001(26)
---------------------------------------------------------------
Séance:         tp_traitement_image_silr1_7 "TP traitement d'image" - tp - logiciel_traitement_image
Date:           9/4 11h30-13h00
Groupes:        silr1
Profs:          gelgon, perreira
Salle:          c001(26)
---------------------------------------------------------------
Séance:         tp_traitement_image_silr2_7 "TP traitement d'image" - tp - logiciel_traitement_image
Date:           9/4 11h30-13h00
Groupes:        silr2
Profs:          marcus
Salle:          b001(26)
---------------------------------------------------------------
Séance:         projet_transversal_id_17 "PROJET projet transversal" - projet - projet_transversal
Date:           9/4 11h30-13h00
Groupes:        id
Profs:          prof_ptrans_id
Salle:          maison_projet(1000)
---------------------------------------------------------------
Séance:         tp_c_silr2_9 "TP C++" - tp - logiciel_c
Date:           9/4 14h00-15h30
Groupes:        silr2
Profs:          picarougne
Salle:          b001(26)
---------------------------------------------------------------
Séance:         ds_bdd "DS BDD" - ds - connaissances_bdd
Date:           10/4 09h45-11h15
Groupes:        info
Profs:          raschia
Salle:          a1(300)
---------------------------------------------------------------
Séance:         tp_c_silr2_10 "TP C++" - tp - logiciel_c
Date:           10/4 11h30-13h00
Groupes:        silr2
Profs:          picarougne
Salle:          c002(26)
---------------------------------------------------------------
Séance:         tp_c_id_9 "TP C++" - tp - logiciel_c
Date:           10/4 11h30-13h00
Groupes:        id
Profs:          soufiane
Salle:          c001(26)
---------------------------------------------------------------
```

FIGURE 5 – Affichage en console

3.4.4 Tests

Les prédicats sont testés à l'aide PLUnit sur notre premier jeu de données `instance_old.pl`. Avoir une démarche TDD (Test Driven Development) nous a grandement aidé à la découpe du problème. Cela nous a également

permit d'arriver rapidement à un code de qualité et de pouvoir travailler à l'amélioration et aux performances de notre solution.

Voici un exemple :

```
:- begin_tests(indiceSeance).

test("indiceSeance renvoie 0 si la seance ne suit rien") :-
    indiceSeance(1, 0).

test("indiceSeance renvoie le nombre de seances precedentes sinon") :-
    indiceSeance(2, 1),
    indiceSeance(8, 2).

:- end_tests(indiceSeance).
```

Listing 13 – Exemple de test

```
swipl -s run_tests.pl
```

Listing 14 – Lancer les tests

3.4.5 Question subsidiaire

L'algorithme `planifier(+Ss, +Ds, -Cs)` permet également de planifier des séances dans un emploi du temps déjà réalisé (`Ds`).

```
findall(S, seance(S, _, _, _), Ss), % toutes les seances
predsort(beforeSeance, Ss, Ss2),
reverse(Ss2, [X|Ss3]), % X est une seance qu'on ne veut pas planifier tout
                       % de suite
planifier(Ss3, Cs), % premier emploi du temps

planifier([X], Cs, Cs2). % ajout de X dans l'emploi du temps si possible
```

Listing 15 – Exemple d'ajout dans un EDT