École Centrale Lyon

Algorithme et raisonnement
BE
Rapport

BE3 - Contraintes

Élèves :
Hugo Puybareau
Mattéo Rousseau

Enseignant :
Emmanuel Dellandrea

18 avril 2024

Table des matières

1 Introduction **2**

2 Planification ***(5 points)*** **2**
2.1 Question 1 3
2.2 Question 2 5

3 Réunions ***(3 points)*** **5**

4 Ordonnancement PERT ***(2 points)*** **8**
4.1 Question 1 8
4.2 Question 2 9

5 Ordonnancement disjonctif ***(2 points)*** **10**

6 Qui a gagné la médaille d'or ? ***(3 points)*** **10**

7 Problème d'emploi du temps ***(2 points)*** **13**
7.1 Question 1 13
7.2 Question 2 14

8 Problème d'emploi du temps ***(5 points)*** **14**

1 Introduction

Dans cette BE sur les contraintes, il nous est demandé de réaliser plusieurs exercices parmi ceux proposés dans l'énoncé.

Le choix est libre à conditions que la somme des points des exercices soit supérieure à 20 et que nous traitions au moins un exercice de 5 points et un exercice de 3 points.

2 Planification *(5 points)*

On construit une maison avec différentes tâches et un budget limité. On veut donc lancer différentes tâches en parallèle en prenant en compte une contrainte de budget minimiser et en réduisant au maximum la durée totale.

On rappelle les notations :

— Dxx : Début tâche xx
— Fxx : Fin tâche xx
— Lxx : Durée tâhce xx
— Cxx : Coût de xx = $1000Dxx$

Enfin, le coût total est de 29000, c'est équivalent à la somme des durées.

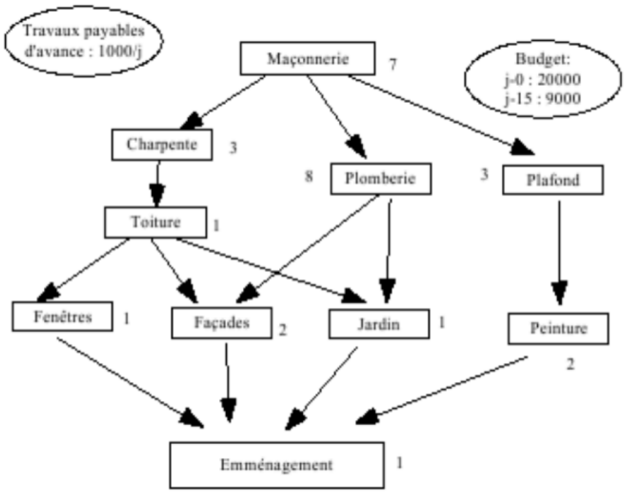

FIGURE 1 – Arbre de succession des tâches

2.1 Question 1

Je crée le code suivant :

```
:- use_module(library(clpfd)).
:- set_prolog_flag(print_depth, 100).

planification(Taches):-
%Variables
Taches = [DM, DC, DT, DPLO, DPLA, DFE, DFA, DJ, DPE, DE, DF],

%Domaines

Taches ins 1..29,
LM#=7, LC#=3, LT#=1, LPLO#=8, LPLA#=3, LFE#=1, LFA#=2, LJ#=1, LPE#=2, LE#=1,

%Contraintes date
DM#=1,
DC#>=DM+LM,
DT#>=DC+LC,
DFE#>=DT+LT,
DFA#>=DT+LT,
DJ#>=DT+LT,
DPLO#>=DM+LM,

DFA#>=DPLO+LPLO,
DJ#>=DPLO+LPLO,

DPLA#>=DM+LM,

DPE#>=DPLA+LPLA,
DE#>=DFE+LFE,
DE#>=DFA+LFA,
DE#>=DJ+LJ,
DE#>=DPE+LPE,
DF#>=DE+LE,

label(Taches).
```

FIGURE 2 – Code pour la planification sans la prise en compte du budget

Pour les test j'obtiens cette sortie :

```
?- planification(L).
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 11|...] ;
L = [1, 8, 11, 8, 8, 12, 16, 16, 12|...] ;
```

FIGURE 3 – Test pour le prédicat *'planification'*

Or la liste ne s'affiche pas en entier. Malgré de nombreuses tentatives d'ajout de ligne dans le code telle que ' *:- set_prolog_flag(answer_write_options, [max_depth(0)]).*', la seule solution que j'ai trouvé pour voir la fin de la liste est de redéfinir *DF* avec uniquement la date d'emménagement et la date de fin :

```
?- planification(L).
L = [18, 19] ;
L = [18, 20] ;
L = [18, 21] ;
L = [18, 22] ;
L = [18, 23] ;
L = [18, 24] ;
L = [18, 25] ;
L = [18, 26] ;
L = [18, 27] ;
L = [18, 28] ;
L = [18, 29] ;
L = [19, 20] ;
L = [19, 21] ;
L = [19, 22] ;
L = [19, 23]
```

FIGURE 4 – Test pour voir la date d'emménagement et la date de fin

C'est bien ce que l'on attend du programme.

2.2 Question 2

Pour la suite, il faut intégrer la contrainte de budget. Il faut qu'avant le jour 15, nous ayons dépensé moins de 20 000 euros. Pour ce faire je code la partie budget la manière suivante :

```
% Limite de budget
Limite = 15,
DM #=< Limite #<==> BDM,
DC #=< Limite #<==> BDC,
DT #=< Limite #<==> BDT,
DPLO #=< Limite #<==> BDPLO,
DPLA #=< Limite #<==> BDPLA,
DFE #=< Limite #<==> BDFE,
DFA #=< Limite #<==> BDFA,
DJ #=< Limite #<==> BDJ,
DPE #=< Limite #<==> BDPE,
DE #=< Limite #<==> BDE,

% Variables de budget qui sont soit 0 soit la durée si la tâche commence avant la limite
Budget15 #= LM*BDM + LC*BDC + LT*BDT + LPLO*BDPLO + LPLA*BDPLA + LFE*BDFE + LFA*BDFA + LJ*BDJ + LPE*BDPE + LE*BDE,
Budget15 #=< 20,

% Instanciation
label(Taches).
```

FIGURE 5 – Code pour la partie budget

En testant, on obtient les sorties suivantes, le budget est bien respecté avec l'arrangement décrit :

```
?- planification(Taches, Budget15).
Taches = [1,8,11,8,16,12,16,16,19,21],
Budget15 = 20 ;
Taches = [1,8,11,8,16,12,16,16,19,22],
Budget15 = 20 ;
```

FIGURE 6 – Test pour la partie sur le budget

3 Réunions *(3 points)*

Nous implémentons la base de données dans PROLOG dans un premier temps :

```
%Base de données

disponible(jean, 540, 600).
disponible(jean, 810, 855).
disponible(jean, 2115, 2145).
disponible(jean, 2340, 2460).
disponible(jean, 3360, 3495).
disponible(jean, 5040, 5070).
disponible(jean, 6360, 6435).
```

FIGURE 7 – Code pour la partie de la base de données pour Jean

Les données numériques sont toutes multipliées par 60 pour traiter des minutes. Il est nécessaire de faire de même pour les disponibilité de Marie, Pierre, Jacques et de la salle.

Pour trouver le créneau disponible, on rédige les deux prédicats suivants :

```
%Je crée le prédicat "agenda" :

agenda_45(Personne, Debut, Fin) :-
    disponible(Personne, H1, H2),
    Debut in H1..H2,
    Fin #= Debut + 45,
    Fin #=< H2.

%Puis le prédicat qui permet de trouver le créneau

reunion(Debut, Fin) :-
    agenda_45(jean, Debut, Fin),
    agenda_45(marie, Debut, Fin),
    agenda_45(pierre, Debut, Fin),
    agenda_45(jacques, Debut, Fin),
    agenda_45(salle, Debut,Fin),
    label([Debut, Fin]).
```

FIGURE 8 – Code pour l'exercice sur les réunions

Puis, après un test, nous trouvons :

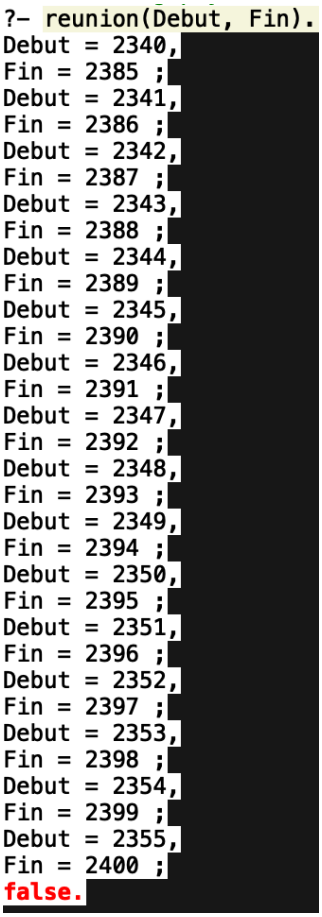

FIGURE 9 – Test pour l'exercice sur les réunions

Ce qui divisé par 60 donne les réunions qui débutent le **Mardi entre 15h et 15h15** et qui terminent le **Mardi entre 15h45 et 16h**.

Après vérification dans l'énoncé, c'est bien ce que nous attendions.

4 Ordonnancement PERT *(2 points)*

4.1 Question 1

Pour la première question, nous devons écrire un programme permettant de trouver la première date de démarrage possible pour la tâche *'J'*.

J'ai donc le code suivant :

```
:- use_module(library(clpfd)).

pert(Taches, StartJ) :-
    Taches = [A, B, C, D, E, F, G, H, I, J],
    Taches ins 0..30, % Définir le domaine des dates de démarrage

    % Contraintes de précédence avec les durées des tâches
    B #>= A + 5,
    C #>= A + 5,
    D #>= A + 5,
    E #>= B + 4,
    F #>= C + 3,
    G #>= C + 3,
    H #>= D + 2,
    I #>= F + 5,
    I #>= G + 4,
    J #>= H + 3,
    J #>= I + 2,

    % Labelliser les variables pour trouver une solution
    labeling([], Taches),

    % J est la dernière tâche, donc sa date de début est la date de début la plus tôt
 pour terminer toutes les tâches
    StartJ = J.
```

FIGURE 10 – Code pour la première question

En testant nous obtenons $\mathbf{J = 15}$, cela semble logique après une vérification à la main :

```
?- pert(Taches, StartJ).
Taches = [0, 5, 5, 5, 9, 8, 8, 7, 13|...],
StartJ = 15
```

FIGURE 11 – Test pour la première question

4.2 Question 2

J'exécute le même code en fixant J et observe les sorties :

```
Taches = [0, 5, 5, 5, 9, 8, 8, 7, 13|...] ;
Taches = [0, 5, 5, 5, 9, 8, 8, 8, 13|...] ;
```

FIGURE 12 – Test pour la deuxième question

```
Taches = [0, 5, 5, 5, 9, 8, 8, 12, 13|...] ;
Taches = [0, 5, 5, 5, 9, 8, 9, 7, 13|...] ;
```

FIGURE 13 – Test pour la deuxième question

```
Taches = [0, 5, 5, 5, 9, 8, 9, 12, 13|...] ;
Taches = [0, 5, 5, 5, 10, 8, 8, 7, 13|...] ;
```

FIGURE 14 – Test pour la deuxième question

Il en est de même pour B et D... On conclu que les taches **B, D, E, G et H** peuvent varier.

5 Ordonnancement disjonctif *(2 points)*

On garde le code précédent et on ajoute les contraintes suivantes qui correspondent à ce qui est dit dans l'énoncé :

```
:- use_module(library(clpfd)).

pert(Taches, StartJ) :-
    Taches = [A, B, C, D, E, F, G, H, I, J],
    Taches ins 0..20, % Définir le domaine des dates de démarrage

    % Contraintes de précédence avec les durées des tâches
    B #>= A + 5,
    C #>= A + 5,
    D #>= A + 5,
    E #>= B + 4,
    F #>= C + 3,
    G #>= C + 3,
    H #>= D + 2,
    I #>= F + 5,
    I #>= G + 4,
    J #>= H + 3,
    J #>= I + 2,

    (   G+4 #=< E; E+1 #=< G),
    (   E+1 #=< F; F+5 #=< E),
    (   F+5 #=< G; G+4 #=< F),

    % Labelliser les variables pour trouver une solution
    labeling([], Taches),

    % J est la dernière tâche, donc sa date de début est la date de début la plus tôt
 pour terminer toutes les tâches
    StartJ = J.
```

FIGURE 15 – Code pour l'ordonnancement disjonctif

Pour les tests, la seule solution que j'ai trouvé est d'observer les variations dans les nombreux retour de la console.

J'ai noté que le marche puisque je n'ai jamais observé de sortie où G,F et E étaient égaux entre eux. De plus, j'observe que **E** peut valoir **12**, puis varie entre **17 et 30**. **F** varie entre **8 et 9** et **G** va de **13 à 14**.

6 Qui a gagné la médaille d'or ? *(3 points)*

On nous donne dans ce problème une liste de contraintes qui doivent nous permettre de trouver le résultat d'une épreuve aux jeux olympiques. Pour commencer, on définit notre objet *Personnes* qui sera l'attribut de réponse du prédicat *medaille*.

Pour effectuer la propagation de contraintes, il faut d'abord définir l'ensemble des contraintes qui nous sont transmises :

```
medaille(Personnes):-
    Personnes = [personne(personne1,_,_,_,_,_),
                 personne(personne2,_,_,_,_,_),
                 personne(personne3,_,_,_,_,_),
                 personne(personne4,_,_,_,_,_),
                 personne(personne5,_,_,_,_,_)],

    Contraintes=
    [
    personne(_, _,  pierre, brascass, _, _),
    personne(_, cinq, _, entorse, _, _),
    personne(_, _, paul, _, equitation, _),
    personne(Paul, _, _, _, equitation, _),
    personne(_, _, _, _, sautaperche, cafe),
    personne(_, _, _, insolation,   _, jus),
    personne(_, _, _, rhume, decathlon, _),
    personne(_, or, _, _, _, eau),
    personne(The, _, andre, _, _, the),
    personne(personne3, _, _, _, escrime, _),
    personne(personne1, _, _, cocardeoeil, _, _),
    personne(Qua, quatre, _, _, _, _),
    personne(Lait, _, _, _, _, lait),
    personne(Bron, bronze, _, _, _, _),
    personne(Jean,  _, jean, _, _, _),
    personne(Cocard, _, _, cocardeoeil, _, _),
    personne(Rolant, _, rolant, _, _, _),
    personne(_, _, _, _, box, _),
    personne(_, argent, _, _, _, _)
    ],
```

FIGURE 16 – Code des contraintes du problème

On peut ensuite définir les contraintes qui sont liées à la géométrie du problème, c'est-à-dire le fait que les personnes sont ordonnées par ordre de récompense à l'épreuve :

```
31      droit(Paul,Rolant),
32      cote(Lait,Qua),
33      cote(The,Bron),
34      cote(The,Bron),
35      cote(Jean,Cocard),
36
37      ajuste(Contraintes,Personnes).
38
39  %definition des contraintes
40
41  droit(personne2,personne1).
42  droit(personne3,personne2).
43  droit(personne4,personne3).
44  droit(personne5,personne4).
45
46  cote(H1,H2) :- droit(H1,H2).
47  cote(H1,H2) :- droit(H2,H1).
48
49  ajuste([],_).
50  ajuste([H|T],List) :-
51      member(H,List),
52      ajuste(T,List).
```

FIGURE 17 – Code des contraintes d'ordre d'arrivée

On tient également compte du fait que toutes les contraintes définies au début du problème doivent être fusionnées pour obtenir 5 participant et le résultat de leur performance.

Lorsque l'on lance la requête, on obtient bien :

```
?- |    medaille(Personnes).
Personnes = [personne(personne1, quatre, andre, cocardeoeil, box, the), personne(pers
onne2, bronze, jean, rhume, decathlon, lait), personne(personne3, or, pierre, brascas
s, escrime, eau), personne(personne4, cinq, rolant, entorse, sautaperche, cafe), pers
onne(personne5, argent, paul, insolation, equitation, jus)] ;
false.
```

FIGURE 18 – Résultat de la requête pour la résolution des contraintes

7 Problème d'emploi du temps *(2 points)*

Les organisateurs d'un congrès disposent de 3 salles et 2 jours pour organiser la tenue d'un congrès de 11 sessions (de A à K) d'une demi-journée.

7.1 Question 1

Certaines sessions ne peuvent pas se dérouler en même temps, ou les unes après les autres. Ces contraintes sont précisées dans l'énoncé et sont intégrées dans le code suivant :

```
:- use_module(library(clpfd)).

emploi_du_temps(Session) :-
    Session = [A, B, C, D, E, F, G, H, I, J, K],
    Session ins 1..4,
%Car 11 sessions pour 12 créneaux au total
%dans 3 salles différentes (3x4=12)
    dif(A,J),
    dif(J,I),
    dif(I,E),
    dif(E,C),
    dif(C,F),
    dif(F,G),
    dif(D,H),
    dif(B,D),
    dif(K,E),

    all_different([B, I, H, G]),
    all_different([A, G, E]),
    all_different([B, H, K]),
    all_different([A, B, C, H]),
    all_different([D, F, J]),

    J #> E,
    K #> D,
    K #> F,

    label(Session).
```

FIGURE 19 – Code pour la question 1

Après un test, on observe que le code fonctionne bien :

```
?- emploi_du_temps(Session).
Session = [1,2,3,1,2,2,3,4,1,3,3] ;
Session = [1,2,3,1,2,2,3,4,1,4,3] ;
Session = [1,2,4,1,2,2,4,3,1,3,4] ;
Session = [1,2,4,1,2,2,4,3,1,4,4] ;
Session = [1,2,4,1,2,3,4,3,1,4,4] ;
Session = [1,2,4,1,3,2,4,3,1,4,4] ;
Session = [1,2,4,1,3,3,4,3,1,4,4] ;
Session = [1,3,4,1,2,2,4,2,1,3,4] ;
Session = [1,3,4,1,2,2,4,2,1,4,4]
```

FIGURE 20 – Test pour la question 1

7.2 Question 2

Il est maintenant demandé de respecter le fait que les sessions ne se déroulent que dans 3 salles différentes.

On intègre cette nouvelle contrainte dans le code de la manière suivante :

```
    % Prédicat pour la répartition des salles
    repartition_salles(Sessions, Salles),

    % Trouver une solution
    label(Sessions).

salle_unique_par_creneau(Associations) :-
    % Créer une liste de créneaux uniques à partir des associations
    findall(Creneau, member(Creneau-_, Associations), Creneaux),
    sort(Creneaux, CreneauxUniques),
    % Appliquer la contrainte pour chaque créneau
    maplist(creneau_a_salle_unique(Associations), CreneauxUniques).

creneau_a_salle_unique(Associations, Creneau) :-
    % Trouver toutes les salles pour un créneau donné
    findall(Salle, member(Creneau-Salle, Associations), Salles),
    % Appliquer la contrainte all_different pour garantir une salle unique par session pour ce créneau
    all_different(Salles).

repartition_salles(Sessions, Salles) :-
    length(Sessions, Length),
    length(Salles, Length),  % Une salle pour chaque session
    Salles ins 1..3,         % 3 salles disponibles

    % Associer chaque créneau avec une salle
    findall(Creneau-Salle, (nth1(Index, Sessions, Creneau), nth1(Index, Salles, Salle)), Associations),
    % Appliquer la contrainte de salle unique par créneau
    salle_unique_par_creneau(Associations).
```

FIGURE 21 – Ajout des contraintes pour les salles

Ce code ne fonctionne pas malgré de nombreuses recherches je n'arrive pas à débloquer la situation. Il y a sûrement une manière plus simple de résoudre cet exercice.

8 Problème d'emploi du temps *(5 points)*

Il est demandé de réaliser un code similaire à l'exercice précédent :

Pour le test :

Par manque de temps, nous n'avons pas pu finir cet exercice.

```
:- use_module(library(clpfd)).

edt(Cours) :-
    Cours = [Extr_Conn , Sys_info , Res_telecom, Big_Data, Stat_Inge, Tr_Anal_MM,
Rech_op, Repr_data, Res_Info, Inge_Objet, Sys_BD, Meth_EDP],
    Cours ins 1..6,

    %On renseigne les contraintes
    Extr_Conn #> 1,
    Tr_Anal_MM #> 1,
    Big_Data #> 1,

    all_different([Extr_Conn, Stat_Inge, Meth_EDP]),
    all_different([Extr_Conn, Rech_op]),
    all_different([Sys_info, Rech_op]),
    all_different([Extr_Conn, Big_Data, Sys_BD]),
    all_different([Sys_info , Big_Data, Sys_BD]),
    all_different([Res_Info, Res_telecom, Repr_data, Sys_info]),

    Stat_Inge = 3,
    Meth_EDP = 6,
    Inge_Objet=5,
    Rech_op=4,
    Extr_Conn=2,

    label(Cours).
```

FIGURE 22 – Code

```
% /Users/hugopuybareau/Documents/Centrale/
?- edt(Cours).
Cours = [2,1,2,3,3,2,4,3,4,5,4,6]
```

FIGURE 23 – Test