USTHB - FEI - Département d'Informatique
Master 2 "Systèmes Informatiques Intelligents" 2017/2018
Date de l'examen : 16/01/2018 <10h15-11h45>

Examen : Programmation par Contraintes
Corrigé

Exercice 1 (7 points)

On considère le CSP binaire discret P=(X,D,C) suivant :

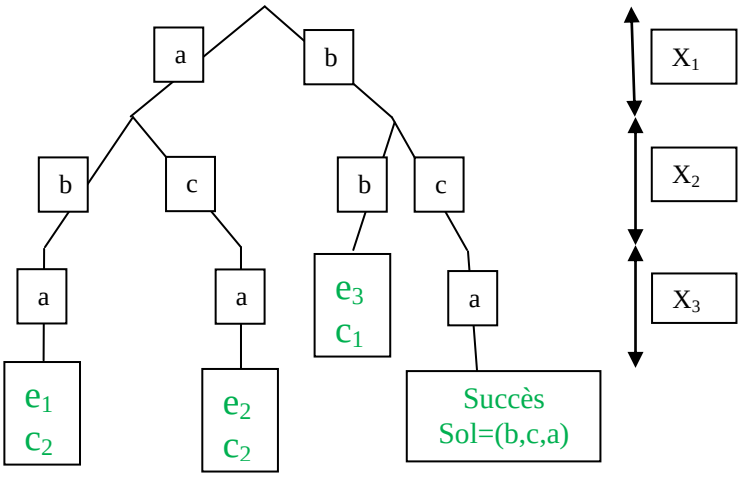
- $X=\{X_1,X_2,X_3\}$
- $D(X_1)=\{a,b,c\}, D(X_2)=\{b,c\}, D(X_3)=\{a\}$
- $C=\{c_1 : X_1 \neq X_2, c_2 : X_1 \neq X_3, c_3 : X_2 \neq X_3\}$

1) Appliquez l'algorithme de recherche SRA (Simple Retour Arrière) au CSP P
2) Appliquez l'algorithme de recherche FC (Forward-Checking) au CSP P
3) Appliquez l'algorithme de recherche Look-Ahead au CSP P

Pour chacun des trois algorithmes de recherche, il est demandé de respecter l'ordre statique X_1, X_2, X_3 d'instanciation des variables ; et l'ordre statique a, b, c de choix des valeurs du domaine de la variable en cours d'instanciation. Il est également demandé d'expliquer tous les échecs éventuels.

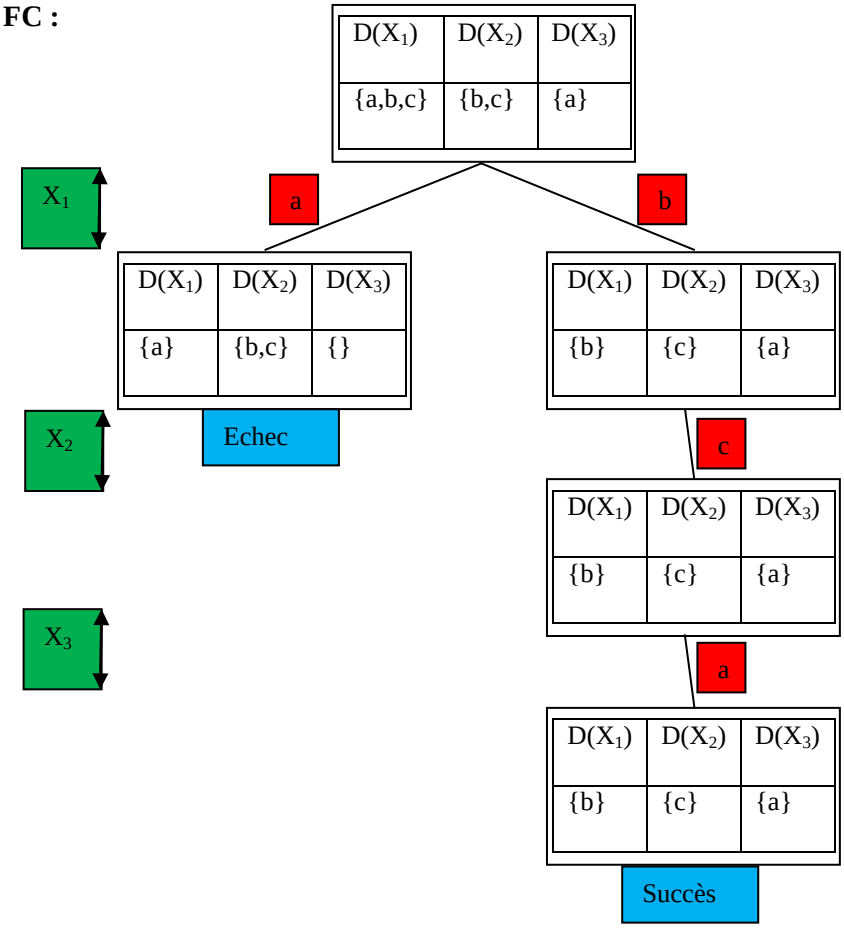
Solution :

1) SRA :



Echecs e_1 et e_2 : la contrainte c_2 entre la variable en cours d'instanciation, X_3, et la variable déjà instanciée X_1 n'est pas satisfaite
Echec e_3 : la contrainte c_1 entre la variable en cours d'instanciation, X_2, et la variable déjà instanciée X_1 n'est pas satisfaite

2) FC :



Explication de l'unique échec :

- **Echec :** L'instanciation de X_1 avec la valeur a de son domaine n'a pas de support dans le domaine de la variable non encore instanciée X_3, et fait passer $D(X_3)$ à l'ensemble vide

3) Look-Ahead :

Look-Ahead commence par un prétraitement du CSP de départ : filtrage avec AC3. Ce prétraitement est fait de la façon suivante :

- La file Q de AC3 est initialisée aux arcs sur lesquels il y a une contrainte, dans les deux sens : $Q=\{(X_1,X_2), (X_2,X_1), (X_1,X_3), (X_3,X_1), (X_2,X_3), (X_3,X_2)\}$. Les domaines, initialement, sont tels que donnés dans le CSP :

$\mathbf{X_i}$	X_1	X_2	X_3
$\mathbf{D(X_i)}$	{a,b,c}	{b,c}	{a}

- On prend l'arc (X_1,X_3) pour propagation. La file devient : $Q=\{(X_1,X_2), (X_2,X_1),(X_3,X_1), (X_2,X_3), (X_3,X_2)\}$. La valeur a de $D(X_1)$, qui n'a pas de support dans $D(X_3)$, est supprimée ; et les domaines deviennent :

$\mathbf{X_i}$	X_1	X_2	X_3
$\mathbf{D(X_i)}$	{b,c}	{b,c}	{a}

- Il n'y a rien à réintroduire dans la file

- On prend maintenant, successivement, les arcs (X_1,X_2), (X_2,X_1), (X_3,X_1), (X_2,X_3) puis (X_3,X_2) pour propagation. On se rend compte, à chaque fois, qu'il n'y a rien à supprimer du domaine de la première variable de la paire prise de la file. Le prétraitement par AC3 prend donc fin sans détecter d'inconsistance, et les domaines en résultant sont donc comme suit :

$\mathbf{X_i}$	X_1	X_2	X_3
$\mathbf{D(X_i)}$	{b,c}	{b,c}	{a}

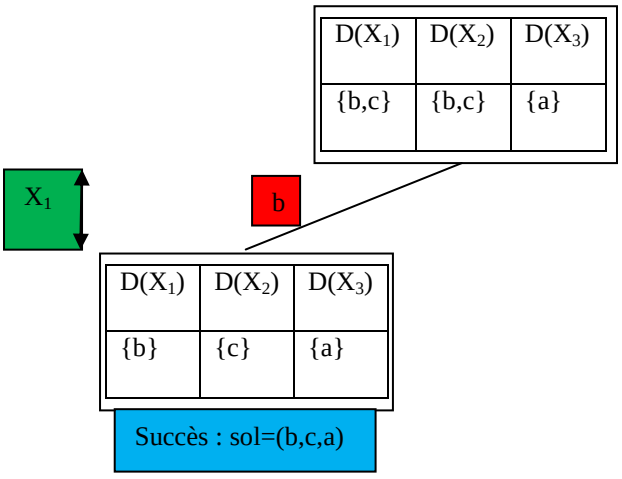
- On instancie la variable X_1 avec la valeur b et on applique le filtrage par AC3 au raffinement résultant dont les domaines sont comme suit :

$\mathbf{X_i}$	X_1	X_2	X_3
$\mathbf{D(X_i)}$	{b}	{b,c}	{a}

- On initialise la file de AC3 aux paires (X_k,X_1) telles qu'il y a une contrainte entre X_k et X_1 : $Q=\{(X_2,X_1),(X_3,X_1)\}$. On prend la paire (X_2,X_1) pour propagation, et la file devient : $Q=\{(X_3,X_1)\}$.
- La valeur b de $D(X_2)$, qui n'a pas de support dans $D(X_1)$, est supprimée ; et les domaines deviennent :

$\mathbf{X_i}$	X_1	X_2	X_3
$\mathbf{D(X_i)}$	{b}	{c}	{a}

- La paire (X_3,X_2) est réintroduite dans la file, qui devient : $Q=\{(X_3,X_1),(X_3,X_2)\}$. On prend ensuite la paire (X_3,X_1) puis la paire (X_3,X_2) pour propagation, et aucune modification n'est à enregistrer.
- Le résultat du filtrage est ainsi un CSP qui, en plus d'être consistant d'arc, est singleton, dans le sens où tous les domaines sont des singletons : on est donc en situation de succès, avec une solution donnée par les domaines : solution=(b,c,a).
- La situation est résumée par l'arbre de recherche suivant, dont les nœuds sont étiquetés par l'évolution des domaines :



Exercice 2 (6 points)
On considère l'instance de la table ci-dessous du problème d'ordonnancement de type job shop, qui consiste en deux jobs J1 et J2 devant passer chacun par deux machines M1 et M2 :

	1ère tâche : <machine, durée>	2ème tâche : <machine, durée>
Job J1	<M2,15>	<M1,18>
Job J2	<M1,5>	<M2,16>

Toutes les tâches sont non-préemptives. La date de début au plus tôt est t_d=3.

On s'intéresse à la recherche d'une solution réalisable, c'est-à-dire satisfaisant toutes les contraintes mais ne donnant pas forcément l'optimum du problème.

1) Modélisez le problème à l'aide d'un TCSP P=(X,C)
2) Donnez la représentation graphique de P
3) Donnez la représentation matricielle de P
4) Donnez **(les grandes lignes d')**un algorithme fournissant, pour un TCSP modélisant une instance du problème d'ordonnancement de type job shop, une solution réalisant l'optimum. Illustrez à l'aide de l'instance en considération

Solution :

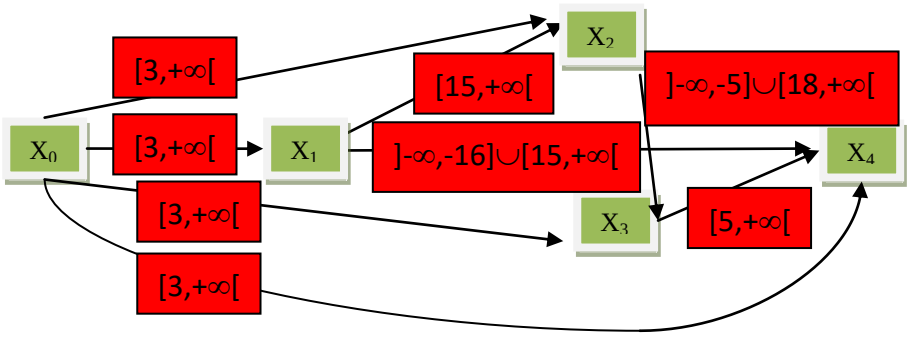
1) **Modélisation à l'aide d'un TCSP :**

L'instance du problème de job shop est constituée de quatre tâches : T_1 (passage du job J1 par la machine M2), T_2 (passage du job J1 par la machine M1), T_3 (passage du job J2 par la machine M1), et T_4 (passage du job J2 par la machine M2). L'instance peut être modélisée par le TCSP P=(X,C) suivant :

- X={ X_0,X_1,X_2,X_3,X_4}, la variable X_0 désignant l'origine du monde, la variable X_i (i de 1 à 4) désignant le début de la tâche T_i
- C={c_1 : $(X_2-X_1)\in[15,+\infty[$,c_2 : $(X_4-X_3)\in[5,+\infty[$,c_3 : $(X_4-X_1)\in]-\infty,-16]\cup[15,+\infty[$, c_4 : $(X_3-X_2)\in]-\infty,-5]\cup[18,+\infty[$, c_5 : $(X_1-X_0)\in[3,+\infty[$, c_6 : $(X_2-X_0)\in[3,+\infty[$, c_7 : $(X_3-X_0)\in[3,+\infty[$, c_8 : $(X_4-X_0)\in[3,+\infty[$}

  - Les contraintes c_1 et c_2 sont des contraintes conjonctives (de précédence) entre les tâches d'un même job ; les contraintes c_3 et c_4 sont des contraintes disjonctives entre les tâches de jobs différents devant passer par une même machine ; et, enfin, les tâches c_5 à c_8 expriment que chacune des quatre tâches a la date t_d=3 comme date de début au plus tôt

2) **Représentation graphique :** Gp=(X,E,I)



3) **Représentation matricielle :**

{0}	[3,+∞[	[3,+∞[	[3,+∞[	[3,+∞[
]-∞,-3]	{0}	[15,+∞[	]-∞,+∞[	]-∞,-16]∪[15,+∞[
]-∞,-3]	]-∞,-15]	{0}	]-∞,-5]∪[18,+∞[	]-∞,+∞[
]-∞,-3]	]-∞,+∞[	]-∞,-18]∪[5,+∞[	{0}	[5,+∞[
]-∞,-3]	]-∞,-15]∪[16,+∞[	]-∞,+∞[	]-∞,-5]	{0}

4) **Adaptation de l'algorithme naïf GET (Générer et Tester) :**

- Générer tous les raffinements convexes maximaux (qui sont donc des STP)
- Pour chaque raffinement Q, calculer son optimum et une solution le réalisant de la façon suivante. Appliquer PC2 à Q : Q=PC2(Q). L'optimum optimum(Q) est maintenant donné par la solution faisant commencer le monde à X_0=0, et instanciant les autres variables X_i,i=1 à N, avec les bornes inférieures bf(i) des entrées [0,i] de la représentation matricielle M de Q :
  - optimum(Q)=$\max_{i=1..N}(b(i)+durée(i))-\min_{i=1..N} b(i)$, durée(i) étant la durée de la tâche T_i
  - sol(Q)=(0,bf(1),…,bf(N))
- Prendre le meilleur des optima des différents raffinements convexes maximaux pour avoir l'optimum du TCSP de départ, le TCSP modélisant l'instance du job shop

Le TCSP modélisant l'instance en considération a, en tout, 4 raffinements convexes maximaux.

L'inconvénient de l'adaptation de l'algorithme naïf GET est, bien évidemment, calculatoire. En effet, chacun des n job devant passer par chacune des m machines, une machine seule donnera lieu à (n-1)+(n-2)+…+1=$\frac{(n-1)n}{2}$ contraintes disjonctives, qui interdiront tout chevauchement à chaque paire des n tâches que la machine doit traiter, une tâche étant vue ici comme le passage d'un job par une machine. Le nombre total de contraintes disjonctives pour une instance de n jobs et m machines, sera donc $\frac{(n-1)nm}{2}$. Le nombre de raffinements convexes maximaux sera donc exponentiel en n et m, égal à $2^{\frac{(n-1)nm}{2}}$.

Adaptation de l'algorithme ''intelligent'' Look-Ahead : Une façon de remédier à l'explosion combinatoire de l'algorithme naïf ci-dessus serait d'adapter l'algorithme Look-Ahead avec, pour éviter le problème dit de fragmentation, un filtrage par un algorithme de consistance de chemin faible tel que PC2 faible ou wPC2 (qui

remplace la composition classique par la composition faible, qui est tout simplement la composition classique des fermetures convexes).

Exercice 3 (7 points) :

On utilise une liste de listes pour représenter la matrice d'adjacence d'un graphe simple G=(V,E) : si N est la taille (nombre de sommets) du graphe, la liste aura N sous-listes toutes de taille N, la $i^{ème}$ sous-liste représentera la $i^{ème}$ ligne de la matrice, le $j^{ème}$ élément de la $i^{ème}$ sous-liste sera l'entrée [i,j] de la matrice. Le but de l'exercice est d'écrire un programme Prolog qui fera appel au solveur de contraintes sur domaines discrets pour générer tous les graphes simples de taille N≥1, et trouver pour chaque graphe généré tous ses cycles hamiltoniens. Pour ce faire, il vous est demandé de procéder comme suit pour la construction des paquets définissant les différents prédicats du programme :

1) Ecrire un paquet définissant le prédicat **liste_de_listes** d'arité 3 suivant : liste_de_listes(M,N,L) réussira si et seulement si L est une liste de M listes toutes de taille N
2) Ecrire un paquet définissant le prédicat **graphe_simple** d'arité 2 suivant : graphe_simple(N,L) réussira si et seulement si L représente un graphe simple de taille N≥1
3) Ecrire un paquet définissant le prédicat **arete** d'arité 3 : arete(X,Y,L) réussira si et seulement si (X,Y) est une arête du graphe simple représenté par L
4) Ecrire un paquet définissant le prédicat **hamiltonien** d'arité 2 suivant : hamiltonien(H,L) réussira si et seulement si H est une liste de taille N d'entiers de 1 à N, tous différents, et H représente un cycle hamiltonien du graphe simple représenté par L (H de la forme $[a_1,\ldots,a_N]$ tels que (a_i,a_{i+1}) arête du graphe, pour tout i de 1 à (N-1), et (a_N,a_1) également arête du graphe)
5) Ecrire un paquet définissant le prédicat **mystere** d'arité 2 : mystere(N,L) réussira si et seulement si la liste de listes L représente un graphe simple de taille N, et le graphe représenté contient un cycle hamiltonien

Solution :

```
:- use_module(library('clp/bounds')).
/*******
1)
*******/
liste_de_listes(1,N,[L]):-length(L,N),!.
liste_de_listes(M,N,[L1|L2]):-length(L1,N),
        M2 is M-1,
        liste_de_listes(M2,N,L2).
/*******
2)
*******/
graphe_simple(N,L):-liste_de_listes(N,N,L), /* vérifie que la matrice représentée par L est carrée, NxN */
        domaine(L),                         /* chaque entrée de la matrice est vue comme une variable dont
                                               le domaine est {0,1} */
        contraintes(L),                     /* génère les contraintes restreignant l'attention aux seules
                                               matrices symétriques,représentant un graphe simple */
        concat(L,L2),                       /* met toutes les variables réparties sur les différentes
                                               sous-listes de L, dans une seule liste, la liste L2 */
        label(L2).                          /* appel du solveur pour génératon d'un graphe simple de taille
                                               N */
domaine([L]):-L in 0..1,!.
domaine([L1|L2]):-domaine([L1]),
        domaine(L2).
```

```
/* les contraintes sont générées de la façon suivante : on démarre avec une matrice NxN. On génère les
   contraintes liant la première ligne et la première colonne : contrainte d'égalité entre la variable
   représentée par le ième élément de la ligne, et la variable représentée par le ième élément de la colonne.
   On passe ensuite à la matrice carrée (N-1)x(N-1) obtenue en supprimant la 1ère ligne et la 1ère colonne, et
   on réitère le processus. Condition d'arrêt (1ère clause du paquet "contraintes") : quand on arrive à la
   matrice carrée 1x1, pour laquelle il n'y a pas de contraintes à générer */
contraintes([[_]]):-!.
contraintes([[X,Y|L1],[Z|L2]|L3]):-contraintes_ligne_colonne([[X,Y|L1],[Z|L2]|L3]),
        supprimer_colonne1([[Z|L2]|L3],L4),
        contraintes(L4).
contraintes_ligne_colonne([[_]]):-!.
contraintes_ligne_colonne([[_,X|L1],[Y|_]|L3]):-X#=Y,
        contraintes_ligne_colonne([[X|L1]|L3]).
concat([],[]):-!.
concat([L],L):-!.
concat([L1,L2|L3],L4):-append(L1,L2,L5),
        concat([L5|L3],L4).
supprimer_colonne1([[_|L1]],[L1]):-!.
supprimer_colonne1([[_|L1]|L2],[L1|L3]):-supprimer_colonne1(L2,L3).
/*******
3)
*******/
arete(1,Y,[L|_]):-egal_1(Y,L),!.
arete(X,Y,[_|L]):-X2 is X-1,
        arete(X2,Y,L).
/* egal_1(Y,L) réussira si et seulement si l'élément d'indice Y de L est égal à 1 */
egal_1(1,[1|_]):-!.
egal_1(Y,[_|L]):-Y2 is Y-1,
        egal_1(Y2,L).
/*******
4)
*******/
hamiltonien([_],_):-!.
hamiltonien([X,Y|L1],L):-dernier([X,Y|L1],Z),
        chaine([X,Y|L1],L),
        arete(Z,X,L).
dernier([X],X):-!.
dernier([_|L],Y):-dernier(L,Y).
chaine([X,Y],L):-arete(X,Y,L),!.
chaine([X,Y,Z|L1],L):-arete(X,Y,L),
        chaine([Y,Z|L1],L).
```

```
/*******
5)
*******/
mystere(N,L):-
        nl, write('=========================================='),
        nl, write('== L graphe simple, H cycle hamiltonien de L=='), nl,
        graphe_simple(N,L),
        /* à ce niveau du programme, un graphe simple de taille N a été généré sous forme d'une solution
           au CSP résultant de l'appel graphe_simple(N,L). Il faut maintenant extraire tous les cycles
           hamiltoniens du graphe simple ainsi généré. L'appel mystere(N,L) ne réussira que si un tel cycle
           existe pour un des graphes simples générés par l'appel graphe_simple(N,L) */
        length(H,N),
        H in 1..N,
        all_different(H),
        /* la contrainte suivante permet de restreindre les N-uplets H vérifiant all_different(H), à ceux
           commençant par le sommet 1 */
        contrainte_sommet_1(H),
        label(H),
        hamiltonien(H,L),
        write('=========================================='), nl,
        write('H='),
        write(H).
contrainte_sommet_1([X|_]):-X#=1.
```

Une exécution du programme :

Ci-dessous une exécution du programme : les premières solutions à la question mystere(4,L) ➔ quatre graphes simples, chacun présentant deux cycles hamiltoniens

```
?- mystere(4,L).
==================================
== L graphe simple, H cycle hamiltonien de L ==
==================================
H=[1,3,2,4]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 0]] ;
==========================================
H=[1,4,2,3]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 0]] ;
==========================================
H=[1,3,2,4]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 1]] ;
==========================================
H=[1,4,2,3]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 1]] ;
==========================================
H=[1,3,2,4]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 0]] ;
==========================================
H=[1,4,2,3]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 0]] ;
==========================================
H=[1,3,2,4]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 1]] ;
==========================================
H=[1,4,2,3]
L = [[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 1]] ;
```