Matière: **Programmation Par Contraintes**

TP 1

Problème du pigeon généralisé

But : On propose une version plus généralisée du problème du pigeon, au lieu d'avoir **n** pigeons et **n-1** nids, nous avons **n** pigeons et **k** nids avec ($2 \leq k \leq n$)

Travail à faire :

1. Utiliser les tableaux en C pour représenter des instances du problème du pigeon, les variables sont stockées dans un tableau de taille **n**.
   Ex : P[i]=j veut dire que le i-ème pigeons est dans le j-ème nid ($1 \leq i \leq n$ et $1 \leq j \leq k$)
   Il n'est pas nécessaire de représenter les domaines pour ce problème ainsi que la contrainte alldifferent.
   Si P[i]=0 veut dire que la variable P[i] n'a pas encore été instanciée.

2. Créer une fonction *alldifferent* qui prend en paramètre une affectation des variables et leur nombre renvoie 1 ou 0 (vrai ou faux) si toutes les variables sont instanciées avec des valeurs toute différentes.
   Ex :
   alldifferent([1,2,3],3) → 1
   alldifferent([1,2,2,4],4)→0

3. Implémenter l'algorithme GET vu en cours pour générer et tester toutes les instanciations possibles des variables du problème. Le programme devra s'arrêter dès qu'une solution est trouvée et l'afficher.

4. Ajouter une option dans les paramètres du main() afin de donner **n**et **k** directement en ligne de commande
   Ex :
   > ./a.out 5 3
   veut dire que je générer le problème des pigeons pour n=5 et k=3 puis le résoudre.

5. Ajouter une option –p (paramètres du main) afin d'avoir la possibilité d'exécuter le programme avec une option –p qui permettra d'afficher les instanciations de variables générées ainsi que les termes (« solution », « pas une solution »)

Ex :
>./a.out3 3 -p
affichera :

P[1]=1 P[2]=1 P[3]=1	pas une solution
P[1]=2 P[2]=1 P[3]=1	pas une solution
P[1]=3 P[2]=1 P[3]=1	pas une solution
P[1]=1 P[2]=2 P[3]=1	pas une solution
P[1]=2 P[2]=2 P[3]=1	pas une solution
P[1]=3 P[2]=2 P[3]=1	solution

Solution trouvée !! : 3 2 1

6. Modifier le programme afin d'ajouter une option –all qui permet d'avoir la possibilité de trouver toutes les solutions (s'il y en a).

7. Ajouter et afficher le temps d'exécution (en seconde) ainsi que le nombre d'instanciation de variables

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire Belhadj Bouchaib Ain Témouchent
Institut des Sciences et de la Technologie
Année universitaire. 2016/2017

3ère année LMD MI (S5)

Matière: **Programmation Par Contraintes**

TP 2

Arc-consistance généralisée de la contrainte globale *alldifferent*

But : écrire un programme qui lit à partir d'un fichier un CSP définie avec une contrainte globale alldifferent.

Travail à faire :

1. Ecrire une procédure qui prend en paramètre un nom de fichier qui contient un CSP définie de la manière suivante :

   -Nombre de variables
   -liste des domaines

   Exemple :

   ```
   3
   1 2
   1 2 3
   1 3
   ```

Qui définit un CSP P à 3 variables V1, V2, V3 avec les domaines D1= {1,2}, D2={1,2,3}, D3={1,3}. On suppose qu'implicitement les variables sont tous liées par une contrainte globale *alldifferent*.

2. Créer une fonction AC_alldifferent qui prend en paramètre le CSP lu à partir du fichier et retourne un CSP arc-consistant pour la contrainte *alldifferent*.

   Exemple :
   AC_alldifferent(P) retournera le CSP P' tel que D'1= {1,2}, D'2={2,3}, D'3={1,3}. On remarque que la valeur 1 du domaine D'2 a été filtrée car incompatible avec la contrainte.

3. Dans le cas où un domaine devient vide on affiche directement « inconsistant ».
4. Le programme devra se lancer avec en paramètre le nom du fichier contenant le CSP.
   Exemple :
   ./a.out csp.txt
   Où csp.txt est le fichier qui contient le csp.
5. Rendre le travail par email.

Centre Uniersitaire d'Ain Témouchent

Projet PPC

Année universitaire 2016/2017

1 Formalisme des CSPs finis discrets

Dans leurs articles fondateurs, Montanari et Mackworth définissent un réseau de contraintes CSP comme un quadruplet $\mathcal{P}(V, D, C, R)$ où $V = \{v_1, v_2, \ldots, v_n\}$ est l'ensemble fini des variables du CSP, $D = \{D_1, D_2, \ldots, D_n\}$ l'ensemble des domaines finis discrets associés aux variables. Le domaine D_i regroupe les différentes valeurs que peut prendre la variable CSP $v_i \in V$, $C = \{c_1, c_2, \ldots, c_n\}$ est l'ensemble des contraintes liant les variables et $R = \{r_1, r_2, \ldots, r_n\}$ l'ensemble des relations correspondant aux contraintes. La relation $r_i \in R$ énumère l'ensemble des tuples permis par la contrainte $c_i \in C$. Pour plus d'informations sur ce sujet, le lecteur peut se référer au livre d'Edward Zhang portant sur les problèmes de contraintes.

Une instantiation du CSP $\mathcal{P}(V, D, C, R)$ est une application I qui affecte une valeur à chaque variable du CSP $\mathcal{P}$ parmi celles figurant dans son domaine. Une contrainte $c_i \in C$ est satisfaite par I si la restriction de I aux variables de la contrainte c_i est un tuple permis par c_i. C'est à dire un tuple de valeurs figurant parmi ceux de la relation correspondante $r_i \in R$. L'instantiation I est une solution du CSP $\mathcal{P}$ si elle satisfait toutes ses contraintes. Le CSP $\mathcal{P}$ est consistant s'il admet au moins une solution, sinon il est inconsistant. Les questions qu'on se pose assez souvent dans les problèmes CSP sont le test de consistance, le calcul d'une ou de toutes les solutions du CSP.

Exemple 1 (Le problème de coloriabilité)
Comment colorier les n sommets d'un graphe $G = (X, E)$ en utilisant m couleurs différentes de façon qu'il n'y ait pas de sommets adjacents de la même couleur. La figure 1 donne le graphe des contraintes du CSP représentant un problème de coloriabilité d'un graphe à quatre sommets et quatre couleurs.

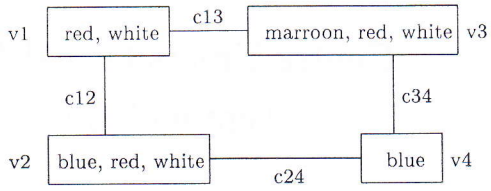

Figure 1: Problème de coloriabilité.

Les variables du CSP $\{v_1, v_2, v_3, v_4\}$ *représentent les sommets du graphe et les couleurs* $\{red, white, blue, marroon\}$ *sont les valeurs des domaines. L'instanciation* $I = (red, white, blue, marroon)$ *est une solution du CSP.*

Plusieurs techniques et algorithmes de résolution des CSPs ont été développés. Les algorithmes de consistance partielle sont nombreux, ils sont en général, basés sur la notion de consistance d'arc, ou celle de consistance de chemin. Ce qui nous intéresse le plus sont les méthodes de consistance globale. Les algorithmes les plus utilisés pour le test de consistance de problèmes CSP sont de type backtrack. On peut en distinguer essentiellement deux catégories : ceux utilisant la stratégie *look ahead* comme *Forward checking* et ceux utilisant la stratégie *look back* comme *Backjumping.*

Parmi plusieurs méthodes testées, celle de *Forward checking* et de *Backjumping* réalisent de bons compromis efficacité et facilité de mise en oeuvre. Nous les avons donc choisies comme nos méthodes de base pour la résolution de CSPs.

2 La méthode Forward checking (notée FC)

Forward Checking est une méthode énumérative du type backtrack. Le principe de cette méthode est basé sur le filtrage des domaines des variables futures (non encore instanciées) par rapport à l'instantiation de la variable courante. Si la variable courante v_i est affectée de la valeur $d_i \in D_i^{\bullet}$, alors on enlève des domaines des autres variables futures liées à v_i par une contrainte toutes les valeurs de leurs domaines qui sont incompatibles avec la valeur d_i de v_i. Si un de ces domaines se vide alors on restaure les domaines par la valeur d_i puis on instancie v_i à une autre valeur de son domaine. Si aucune valeur n'est consistante pour instancier v_i, la méthode effectue alors un rebroussement chronologique vers la variable v_{i-1}. Dans les autres cas, on choisit la prochaine variable à instancier. Le processus est répété

jusqu'à l'obtention d'une solution ou d'une preuve de l'inconsistance (backtrack jusqu'à la racine et plus de valeurs a tester).

Nous donnons dans La figure 2 une description récursive de la méthode FC, où D représente l'ensemble des domaines, I l'interprétation partielle ($I = \emptyset$ au départ) et k l'indice de la variable courante.

```
Procedure FC(D, I, k); { I = [d_1, d_2, ..., d_k]}
begin
  if k = n then [d_1, d_2, ..., d_k] is a solution
  else
  begin
    for each v_i ∈ V, such that C_ik ∈ C, v_i ∈ future(v_k) do
      for each value d_i ∈ D_i do
        if (d_i, d_k) ∉ r_ik then
          delete d_i from D_i;
    if ∀v_i ∈ future(v_k), D_i ≠ ∅ then
    begin
      v_{k+1}=next-variable(v_k)
      repeat
        take d_{k+1} ∈ D_{k+1}
        D_{k+1} = D_{k+1} − d_{k+1}
        I = [d_1, d_2, ..., d_k, d_{k+1}];
        FC(D, I, k + 1);
      untill D_{k+1} = ∅
    end
  end
end;
```

Figure 2: La méthode Forward checking

2.1 Heuristique

Une des meilleurs heuristiques de choix de variables que nous avons trouvée dans la litérature pour FC est celle qui choisit à noeud de l'arbre de recherche durant le processus de résolution, la variable ayant un nombre minimal de valeurs dans son domaine comme prochaine variable à instancier. Cette heuristique est connue sous le nom *Minimum domain heuristic* (MD).

3 La méthode Backtrack (notée BT)

La méthode BT est la méthode $Backtrack$ classique qui consiste à étendre progressivemet une affectation consistante. En cas d'échec, on change la valeur de la variable courante. S'il n'y a plus de valeur, on revient sur la variable précédente. On réitère le procédé tant qu'on n'a pas trouvé une solution et qu'on n'a pas essayé toutes les possibilités. Nous donnons ci-dessous une version récursive l'algorithme $Backtrack$.

```
BT(A, V)
Si V = ∅ Alors A est une solution
Sinon
  Choisir x ∈ V
  d ← d_x
  TantQue d ≠ ∅
      Choisir v dans d
      d ← d\{v}
      Si A ∪ {x ← v} est consistante
      Alors BT(A ∪ {x ← v}, V\{x})
      FinSi
  FinTantQue
FinSi
```

Figure 3: La méthode de base Backtracking

4 Travail à réaliser

Pour des raisons d'efficacité, tous les algorithmes de résolution seront implémentés de façon itérative. Le travail qui vous est demandé est le suivant :

1. Implémenter les algorithmes Backtracking BT avec et sans l'heuristique *MD*.

2. Implémenter une version de l'algorithme Forward Checking (FC) avec et sans l'heuristique *MD*.

4. Application aux problèmes de coloration de graphes générés aléatoirement. Comparer l'efficacité de BT, FC avec et sans heuristiques en termes de temps, de nombre de noeuds. Le générateur d'instances du problème de coloration de graphes doit être programmé. On testera pour n=7 et une densité de 0.5 (densité = nombre d'arête/nombre totale d'arêtes). Les problèmes sont générés en fixant les paramètres suivants : n le nombre de sommets (les variables); NbColors le nombre de couleurs (les valeurs des domaines) et d la densité du graphe des contraintes. Pour n, NbColors et d fixés, le nombre de noeuds NbNodes ainsi que le temps d'exécution Time sont calculés sur la moyenne des résultats obtenus à partir de 100 instances générées aléatoirement.

5 Rendu du travail

En plus d'une vérification sur machine, vous devrez fournir les codes sources en C des différents algorithmes ainsi qu'un rapport. Ce rapport devra con-tenir :

1. Un manuel d'utilisation de vos programmes.

2. Les algorithmes BT, FC avec et sans heuris-tiques.

3. Les résultats et/ou courbes de comparaisons entre les différentes versions avec et sans heuristiques.

4. Les résultats et/ou courbes de comparaisons entre une même méthode (FC ou BJ) avec et sans heuristiques.

Matière: **Programmation Par Contraintes** **(durée : 1h30)**

Examen

Les documents ne sont pas autorisés.
La note dépend du choix de la méthode, du raisonnement et de la clarté de la copie donc soignez vos réponses !!

Rappel : Les 2 expressions suivantes sont équivalentes :
(1) $p \Longrightarrow q$
(2) $\bar{p} \vee q$

1. Ex1 (8 Pts)

"Les Anderson vont nous rendre visite ce soir", annonce Monsieur Blum. "Toute la famille, donc Monsieur et Madame Anderson et leurs trois fils Antoine, Bernard et Claude ?", demande Madame Blum craintive. Monsieur Blum, qui ne rate pas une occasion de provoquer sa femme : "Non, pas du tout. Je t'explique. Si le père Anderson vient, alors il emmène aussi sa femme. Au moins un des deux fils Claude et Bernard vient. Soit Madame Anderson vient, soit Antoine vient. Soit Antoine et Bernard viennent tous les deux, soit ils ne viennent pas. Et si Claude vient, alors Bernard et Monsieur Anderson aussi."

1.1. Formaliser le CSP correspondant au problème.
1.2. Résoudre le CSP et répondre à la question : qui vient donc ce soir ?

2. Ex2 (4.5 Pts)

Soit un problème de satisfaction de contrainte (CSP) sur 5 variables $x_1, \dots, x_5$ de domaine $\{1, 2, 3, 4, 5\}$ chacune. Les contraintes sont

$$\begin{cases} x_3 \geq 4 \\ |x_1 - x_2| = 1 \\ |x_2 - x_3| = 1 \\ \forall i, j \ \{1, \dots, 5\} \ i \neq j : x_i \neq x_j \end{cases}$$

2.1. Rendre le CSP arc-consistant.
2.2. On ajoute la contrainte $x_2 \neq 3$. Y-a-t-il des nouvelles simplifications ?
2.3. Maintenant considérons le fait qu'on remplace les contraintes binaires de différence «$\forall i, j \ \{1, \dots, 5\} \ i \neq j : x_i \neq x_j$ » par $alldifferent(x_1, x_2, x_3, x_4, x_5)$. Y-a-t-il de nouvelles simplifications si on utilise l'arc-consistance généralisée pour les *alldifferents* ?

3. Ex3 (7.5 Pts)

Soit le CSP définit par :
L'ensemble de contrainte $C = \{X + Y = 10,\ X < 8,\ Y > 4\}$
Les domaines des variables X et Y définies par $D_X = [1..10]$; $D_Y = [1..15]$

3.1. Rendre le CSP arc-consistant.
3.2. Donner la micro-structure du CSP.
3.3. Le CSP est-il consistant, si oui donner l'ensemble des solutions.
3.4. On ajoute maintenant la contrainte suivante à C:

"X est premier" $\Longrightarrow$ "Y est premier"

-Donner alors l'ensemble des solutions du CSP.

Bonne chance