176

Initiation à la Programmation Logique par Contraintes

Problèmes et contraintes

177

- Les « problèmes contraints » sont naturellement présents dans notre vie quotidienne :
  - affecter des stages à des étudiants en respectant leurs souhaits ;
  - ranger des pièces de formes diverses dans une boîte rigide ;
  - planifier le trafic aérien pour que tous les avions puissent décoller et atterrir sans se percuter ;
  - ordonnancer des tâches en respectant des délais ;
  - établir un menu à la fois équilibré et appétissant.

Les CSP

178

- La Programmation par Contraintes permet de résoudre un champs bien défini de problèmes :
- La notion de **Problème de Satisfaction de Contraintes** (CSP: *Constraint Satisfaction Problem*) désigne l'ensemble des problèmes constitué par un ensemble de variables et par un ensemble de contraintes sur les valeurs de ces dernières.
- On peut également fournir pour chaque variable un ensemble de valeurs possibles appelé domaine de la variable.
- Résoudre un CSP, consiste à chercher une valeur pour chacune des variables respectant les contraintes.

Les domaines finis

179

- Les domaines finis servent à modéliser les problèmes combinatoires discrets. Ces problèmes peuvent être vus comme des problèmes de recherche dans un espace fini d'un ou plusieurs points.
- Une variable à domaine fini est une variable qui prend sa valeur dans un ensemble fini de constantes ou d'entiers naturels.

Les CSP

180

- Les **CSP** sont généralement combinatoires :
  - il faut souvent envisager un très grand nombre de combinaisons avant d'en trouver une qui satisfasse toutes les contraintes ;
  - bien souvent, la puissance de calcul des ordinateurs ne suffit pas pour examiner toutes les combinaisons possibles en un temps acceptable ;
  - il est nécessaire d'introduire des « **raisonnements** » et des « **heuristiques** » permettant de réduire la combinatoire et de guider la recherche vers les **solutions**.

La programmation par contraintes

181

- La supériorité de la machine vient de sa rapidité pour examiner un grand nombre de cas ;
- La supériorité de l'humain vient de sa capacité à éliminer un grand nombre de cas dès le départ ;
- La **Programmation par Contraintes** tente de trouver un compromis :
  - Utiliser la rapidité de la machine pour examiner un grand nombre de cas ;
  - Utiliser la machine pour exécuter des algorithmes qui permettent de ne pas explorer des cas non intéressants .

La programmation par contraintes

182

- La **Programmation par Contraintes** va permettre :
  - D'exprimer formellement (avec un langage dédié) un problème (les inconnus et les contraintes);
  - De trouver des solutions qui satisfont ces contraintes.
- Une architecture à 2 niveaux :
  - La **Programmation par Contraintes** permet de réduire l'effort de programmation nécessaire dans la spécification du problème et dans l'exécution des programmes :
  - C'est à la fois :
    - Un outils de modélisation ;
    - Un outils de résolution.

Solveurs de Contraintes

183

- Les algorithmes permettant résoudre des CSP sont appelés des « solveurs » de contraintes.
- Certains de ces solveurs ont été intégrés dans des langages de programmation, définissant un nouveau paradigme de programmation appelé **programmation par contraintes** :
  - il suffit de spécifier les contraintes, leur résolution étant prise en charge automatiquement (sans avoir besoin de le programmer) par les solveurs de contraintes intégrés au langage.

Résoudre les problèmes d'optimisation

184

- Programmation Mathématique (analyse numérique)
  - Problèmes continus, fortes hypothèses (convexité, etc.)
- Programmation linéaire
  - Problèmes linéaires
- Meta-heuristiques (recuit simulé, recherches tabou, algorithmes génétiques)
  - Pas de garantie de performance
- **Programmation par contraintes**

Champs d'application

185

- **Contraintes sur des réels**
  - Linéaires : SIMPLEX
  - Non-linéaires : Optimisation non linéaire
- **Contraintes sur des entiers**
  - Linéaires : PLNE (Simplex + B&B)
- Contraintes sur des domaines finis
  - Contraintes logiques
  - Contraintes numériques
  - Contraintes ensembliste ….

PPC ≠ (PL, PLNE)

Programmation par contraintes

186

- **La programmation par contraintes est orthogonal aux autres paradigmes de programmation :**
  - la programmation impérative,
  - la programmation fonctionnelle,
  - la programmation déclarative,
  - la programmation orientée objet :
- **Il existe des langages de programmation par contraintes**
  - logiques (déclaratifs) (CHIP, Eclipse, Prolog IV, Gnu-Prolog),
  - fonctionnels orientés objet (Mozart, Screamer),
  - impératifs orientés objet (CHOCO, Ilog Solver)

PPC : avantages

187

- Langage déclaratif
  - Dans un premier temps on se concentre sur la description du problème sans se préoccuper de sa résolution
- Permet l'intégration de méthodes coopératives
  - Méthodes unifiées dédiées à l'intégration d'algorithmes spécifiques
- Applications
  - Succès prouvé (ex, Planification/Ordonnancement).

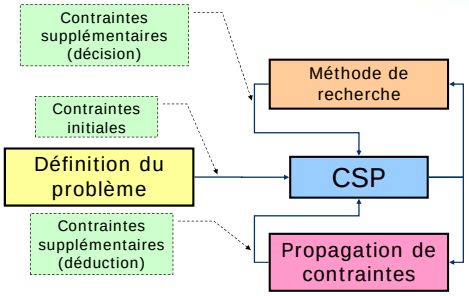
PPC *vs.* RO

188

Programmation par Contraintes	Recherche Opérationnelle
✓ Très flexible ✓ Modélisation facile ✓ Réutilisable ✓ Incapacité à résoudre de grands problèmes (avec des techniques standards)	✓ Algorithmes efficaces (polynomiaux) pour beaucoup de problèmes ✓ Techniques puissantes de bornes inférieures ✓ Modélisation non évidente ✓ Peu flexible (souvent non réutilisable)

La programmation par Contraintes

189



Programmation par contraintes : exemple

190

- $X \in \{1, ..., 10\}$, $Y \in \{2, ..., 20\}$, $Z \in \{3, ..., 10\}$
- $X + Y = 5$, (C_1)
- $Z \neq X + 6$, (C_2)
- $Z > 10 * X - Y$ (C_3)

Propagation :

(C_1) $X \geq 5 - \max(Y) \geq 5 - 20$, $X \leq 5 - \min(Y) \leq 3$	$X \in \{1, 2, 3\}$
(C_1) $Y \geq 5 - \max(X) \geq 5 - 3$, $Y \leq 5 - \min(X) \leq 4$	$Y \in \{2, 3, 4\}$
(C_3) $Z > 10 * \min(X) - \max(Y) > 10 - 4$	$Z \in \{7, ..., 10\}$
(C_3) $Y > 10 * \min(X) - \max(Z) > 10 - 10$	
(C_3) $X < (\max(Z) + \max(Y)) / 10 < (10 + 4) / 10$	$X = 1$ Bound
(C_1) $Y = 5 - 1 = 4$	$Y = 4$ Bound
(C_2) $Z \neq 7$	$Z \in \{8, 9, 10\}$

exemple (ILOG CP)

191

```
#include <ilsolver/ilosolverint.h>
#include <iostream>
void main(){
   IloEnv env;
   IloModel model(env);
   // declarer les variables du problème
   IloIntVar X(env,1,10); IloIntVar Y(env,2,20);
   IloIntVar Z(env,3,10);
   // ajouter les contraintes
   model.add(X+Y==5); model.add(Z!=X+6); model.add(Z>10*X-Y);
   // résoudre le problème
   IloSolver solver(model);
   IloGoal goal=IloAnd(IloInstantiate(X),IloInstantiate(Y),
                       IloInstantiate(Z));
   // résoudre le problème
   if (solver.nextSolution())
        std::cout<<"solution:\n"
                 <<"X="<<X<<" Y="<<Y<<" Z="<<Z<<"\n";
}
```


exemple (Gnu Prolog)

192

```
//ppc_ex1.pl
solve([X,Y,Z]):-
    % declaration des variables avec leur domaines
    fd_domain(X,1,10),
    fd_domain(Y,2,20),
    fd_domain(Z,3,10),
    % expression des contraintes
    X+Y #=# 5,
    Z #\=# X+6,
    Z #># 10*X-Y,
    % résoudre le problème
    fd_labeling([X,Y,Z]).
```


exemple (Gnu Prolog)

193

```
send:-I=[S,E,N,D,M,O,R,Y],
      fd_domain(I,0,9), fd_all_different(I),
      fd_domain([R1,R2,R3,R4],0,1),
      D+E #= Y+10*R1,
      R1+N+R #= E+10*R2,
      R2+E+O #= N+10*R3,
      R3+S+M #= O+10*R4,
      R4 #= M, M #\=0, S #\=0,
      fd_labeling(I), write(I).
```

exemple (Gnu Prolog)

194

```
monnaie(T,P,E2,E1,C50,C20,C10):-
   R is (T-P),
   fd_domain(RE2,0,E2),fd_domain(RE1,0,E1),
   fd_domain(RC50,0,C50),fd_domain(RC20,0,C20),
   fd_domain(RC10,0,C10),
   RE2*200+RE1*100+RC50*50+RC20*20+RC10*10 #= R,
   N #= RE2+RE1+RC50+RC20+RC10,
   fd_minimize(fd_labeling([RE2,RE1,RC50,RC20,RC10]),N),
   write('N='),write(N),write(' Solution='),
   write([RE2,RE1,RC50,RC20,RC10]),nl.
```


Les CSP

195

- La résolution pratique des CSP fait appel à des techniques issues traditionnellement
  - de l'intelligence artificielle
    - formalisme
    - déduction
    - induction …
  - de la Recherche Opérationnelle
    - graphes et leurs outils
    - mathématiques

Types d'applications

196

- Gestion du temps : gestion d'emploi du temps, planification d'équipes, de rotation d'équipes ;
- Gestion et affectation des ressources : gestion du personnel, de moyens de transports, gestion de production ;
- Planification et ordonnancement : planification de production, de livraison, de maintenance, d'interventions, d'itinéraires, ordonnancement d'ateliers ;
- Optimisation : optimisation de production, de moyens, d'investissements, de placements financiers, de coûts.

Types d'applications

197

- Gestion de la cohérence d'information : interfaces graphiques, édition électronique, tableurs, bases de données ;
- Modélisation : modèles mixtes numériques/symboliques, géométriques, économiques, financiers ;
- Simulation comportementale de systèmes ;
- Vérification de spécifications ;

Bibliographie

198

- Quelques supports de cours
  - **Foundations of Constraint Satisfaction,** R. Bartak - 2002
    - http://ktilinux.ms.mff.cuni.cz/~bartak/ESSLLI2002/lectures.html
  - **Programmation Logique par contraintes,** F. Fages - 1996
    - Ed. Ellipses, ISBN 2-7298-4613-1.
- Quelques sites
  - **GNU-PROLOG** : http://gprolog.inria.fr/
  - **ILOG** : http://www-01.ibm.com/software/fr/websphere/ilog/

Qu'est ce qu'une Contrainte ?

199

- Une contrainte est une **relation logique** (une propriété qui doit être vérifiée) entre différentes inconnues, appelées **variables**.

Exemples :

- $X+Y = 3$
- $C_1 \neq C_2$
- $C_1 \neq$ bleu

Qu'est ce qu'une Contrainte ?

200

- Une contrainte **restreint** les valeurs que peuvent prendre simultanément les variables. Par exemple, la contrainte x + 3*y = 12 restreint les valeurs que l'on peut affecter simultanément aux variables x et y.
- Une contrainte est **relationnelle** : elle n'est pas "dirigée". Par ex, x - 2*y = z permet de déterminer z dès lors que x et y sont connues, mais aussi x dès lors que *y* et z sont connues et *y* dès lors que x et z sont connues.
- Une contrainte est **déclarative** : elle spécifie quelle relation on doit retrouver entre les variables, sans donner de procédure opérationnelle pour effectivement assurer/vérifier cette relation.

Qu'est que qu'une Contrainte ?

201

- L'**ordre** dans lequel sont posées les **contraintes** n'est **pas significatif** ;
- la seule chose importante à la fin est que la conjonction de toutes les contraintes soit satisfaite ;
- Cependant, dans certains langages de programmation par contraintes l'ordre dans lequel les contraintes sont ajoutées peut avoir une influence sur l'efficacité de la résolution.

Contraintes

202

Une contrainte peut être **définie** :

- En **intension** ***(en compréhension)*** à l'aide de propriétés mathématiques connues
  - par exemple : $x < y$ ou encore *A et B => non(C)*
- En **extension**, en énumérant les tuples de valeurs appartenant à la relation
  - $x \in [0..2]$, $y \in [0..2]$, $x < y$ devient en extension
    $(x=0 ; y=1)$ *ou* $(x=0 ; y=2)$ *ou* $(x=1 ; y=2)$,
    ou encore $(x,y) \in \{(0,1),(0,2),(1,2)\}$.

Contraintes

203

- Une contrainte est **globale** si elle met en relation un ensemble de variables.
  - atmost (2 ; [X1 ; X2 ; X3 ; X4 ; X5] ; 1) :
    Au plus 2 variables parmi {X1 ; X2 ; X3 ; X4 ; X5} sont égales à 1
  - alldiff ([X1 ; X2 ; X3 ; X4 ; X5]) :
    Les variables { X1 ; X2 ; X3 ; X4 ; X5 } ont des valeurs différentes deux à deux

Contraintes numériques

204

- Une contrainte est numérique si elle porte sur des variables à valeurs numériques
- Une contrainte numérique est :
  - une contrainte d'égalité (=) ,
  - une contrainte d'inégalité ($\neq$),
  - une inéquation ($<$, $\leq$, $>$, $\geq$),

  entre 2 expressions arithmétiques.
- On distingue :
  - les contraintes numériques sur les réels,
  - les contraintes numériques sur les entiers.

Contraintes numériques

205

On distingue également :

- **contraintes numériques linéaires**,
  - quand les expressions arithmétiques sont linéaires
    - par exemple $4*x - 3*y + 8*z < 10$;
- **contraintes numériques non linéaires** ,
  - quand les expressions arithmétiques contiennent des produits de variables, ou des fonctions logarithmiques, exponentielles...
    - par exemple $x*x = 2$ ou $\sin(x) + z*\log(y) = 4$.

Les contraintes booléennes

206

- Une **contraintes est booléenne**, si elle porte sur des variables à valeur booléenne : vrai ou faux.
- Une contrainte booléenne est
  - une implication (=>),
  - une équivalence (<=>)
  - une non équivalence (<≠>)

  entre 2 expressions logiques
- Exemples :
  - (non *a*) ou *b* => *c*
  - non (*a* ou *b*) <=> (*c* et *d*).

Problème de satisfaction de contraintes : formalisation

207

- Un **Problème de Satisfaction de Contraintes** ou **CSP** (Constraint Satisfaction Problem) est un problème modélisé sous la forme :

  (1) un ensemble de **variables** $X_1, \ldots, X_n$
  (2) pour chaque variable X_i, un **domaine de valeurs possibles**
  (3) un ensemble de **contraintes** $C_1, \ldots, C_k$ entre ces variables

- **Question :**
  Existe-t-il une affectation pour chaque variable qui satisfasse toutes les contraintes ? Trouver une telle affectation.
  (problème de décision / problème de recherche)

CSP

208

On définira donc un CSP par un triplet **(*X,D,C*)** tel que :

- ***X* = { X_1, X_2, ..., X_n}** est l'ensemble des variables (les inconnues) du problème ;
- ***D*** est la fonction qui associe à chaque variable X_i son domaine $D(X_i)$, c'est-à-dire l'ensemble des valeurs que peut prendre X_i ;
- ***C* = {C_1, C_2, ..., C_k}** est l'ensemble des contraintes.
  Chaque contrainte C_j est une relation entre certaines variables X, restreignant les valeurs que peuvent prendre simultanément ces variables.

CSP

209

- Par exemple, on peut définir le CSP **(*X, D, C*)** suivant :
  - ***X* = { *a, b, c, d* }**
  - ***D*(*a*) = *D*(*b*) = *D*(*c*) = *D*(*d*) = { 0, 1 }**
  - ***C* = { *a* ≠ *b*, *c* ≠ *d*, *a*+*c* < *b* }**

Affectation

210

On appelle **affectation** le fait d'instancier certaines variables par des valeurs (prises dans les domaines des variables).
- On notera ***A* = { (X_1,V_1), (X_2,V_2), ..., (X_r,V_r) }** l'affectation qui instancie la variable X_1 par la valeur V_1, la variable X_2 par la valeur V_2, ..., et la variable X_r par la valeur V_r.
- Par exemple, sur le CSP, **{(*b*,0),(*c*,1)}** est l'affectation qui instancie ***b*** à 0 et ***c*** à 1.

- Une affectation est **totale** si elle instancie toutes les variables du problème.
- Une affectation est dite **partielle** si elle n'en instancie qu'une partie.

A_1 = {(*a*,1),(*b*,0),(*c*,0),(*d*,0)} est une affectation totale ;
A_2 = {(*a*,0),(*b*,0)} est une affectation partielle.

Violation contraintes

211

- Une affectation (partielle ou totale) ***A* viole** une contrainte C_k si toutes les variables de C_k sont instanciées dans ***A***, et si la relation définie par C_k n'est pas vérifiée pour les valeurs des variables C_k définies dans ***A***.
  - l'affectation partielle A_2 = {(*a*,0),(*b*,0)} viole la contrainte ***a*≠*b*** ;
  - en revanche, elle ne viole pas les deux autres contraintes dans la mesure où certaines de leurs variables ne sont pas instanciées dans A_2.

Consistance

212

- Une affectation (totale ou partielle) est **consistante** si elle ne viole aucune contrainte ;
- Une **affectation** est **inconsistante** si elle viole une ou plusieurs contraintes.
  - l'affectation partielle **{(*c*,0),(*d*,1)}** est consistante, l'affectation partielle **{(*a*,0),(*b*,0)}** est inconsistante ;
- Une **solution** est une affectation totale consistante, *i.e.*, une instanciation de toutes les variables qui ne viole aucune contrainte.
  - l'affectation totale consistante ***A* = {(*a*,0),(*b*,1),(*c*,0),(*d*,1)}** est une solution.

Décision vs. Optimisation

213

- Jusqu'alors, on a vu que la programmation par contraintes ne pouvait résoudre que des **problèmes de décision / de recherche** :
  - Existe-t-il une affectation pour chaque variable (une solution) vérifiant toutes les contraintes du problème ? Donner une telle affectation si elle existe.
- Qu'en est-il des **problèmes d'optimisation** ? :

  Soit une fonction objectif ***F***, qui associe une valeur numérique à chaque solution :
  - Trouver (si elle existe) la solution qui minimise (ou qui maximise) la fonction ***F***.

Résoudre les problèmes d'Optimisation avec la PPC

214

- On ajoute une variable globale ***F*** au CSP qui est contrainte à être égale à la fonction objectif (fonction des autres variables du problèmes).
- On résout alors plusieurs problèmes variantes du problèmes de décision.

Comment résoudre un CSP ?

215

3 approches :

- Énumérer des solutions qui satisfont les contraintes (méthodes d'énumération "Generate and Test" et "Backtracking".
- Utiliser des techniques de cohérence qui réduisent les domaines des variables jusqu'à ce qu'elles soient toutes instanciées.
- Un mélange des deux approches (techniques d'énumération qui utilise des techniques de cohérences).

Résolution CSP : techniques de cohérences

216

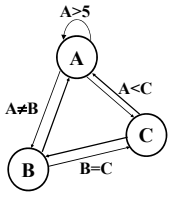
- Méthodes de Consistance d'arc
- Méthodes de Consistance de chemin

Graphe de contraintes associé au CSP

217

- noeuds = variables
- arcs = contraintes

- Variables
  - A, B, C
- Contraintes
  - A ≠ B
  - A > 5
  - A < C
  - B = C



Utiliser les contraintes

218

- Jusqu'ici les contraintes ont été utilisées « passivement »
  - Comme un test ;
  - Au mieux elle servent à détecter d'ou vient le conflit.

Exemple : A, B, $D(A)$=[3..7], $D(B)$=[1..5], $A<B$

Des valeurs « **inconsistantes** » peuvent être retirées des domaines :
$D(A)$=[3..4] et $D(B)$=[4..5]

Comment limiter au mieux les domaines des variables ?

Techniques de cohérence : propagation de contraintes

219

- Les techniques de "cohérence" tentent d'exclure les valeurs incohérentes (inconsistantes) des domaines des variables en utilisant les contraintes.
- La réduction de ces domaines peut mener à une solution.
- Ces techniques permettent de maintenir la cohérence entre les différentes variables à travers le graphe de contraintes : on parle alors de « consistance par arcs » ou « d'arc-consistance ».

Consistance

220

1. Consistance de nœud ;
2. Consistance d'arc ;
3. Consistance de Chemin.

Consistance de Nœud (NC)

221

Node Consistency : un CSP (X, D, C) est **consistant de noeud** si pour toute variable X_i de X, et pour toute valeur v de $D(X_i)$, l'affectation partielle $\{(X_i,v)\}$ satisfait toutes les contraintes **unaires** de C.

Exemple :

Si C contient la contrainte $X_1 > 2$, et $D(X_1)$={1,2,3,4,5},

alors le CSP n'est pas consistant de noeud.

Pour qu'il soit consistant de noeud, il faut enlever du domaine de X_1 les valeurs 1 et 2 qui violent la contrainte $X_1 > 2$.

Consistance d'Arc (AC) : CSP binaires

222

Arc Consistency : un CSP (X, D, C), où C ne contient que des contraintes binaires, est **consistant d'arc** si pour tout couple de variables (X_i, X_k) de X, et pour toute valeur $v \in D(X_i)$, il existe une valeur w appartenant à $D(X_k)$ telle que l'affectation partielle $\{(X_i,v),(X_k,w)\}$ satisfasse toutes les contraintes **binaires** de C.

Exemple : si C contient la contrainte $X_1 + X_2 > 2$, et si $D(X_1)$=$D(X_2)$={0, 1, 2}, alors le CSP n'est pas consistant d'arc car lorsque X_1=0, il n'y a aucune valeur de $D(X_2)$ qui puisse satisfaire la contrainte $X_1 + X_2 > 2$.

Pour qu'il soit consistant d'arc, il faut enlever des domaines de X_1 et X_2 la valeur 0.

Consistance d'Arc (AC) : CSP binaires

223

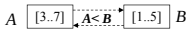
- Dans le cas de CSP à contraintes binaires
  - Une contrainte est un arc
  - L'arc (X_i, X_k) est consistant ssi pour chaque valeur v de $D(X_i)$ il existe une valeur w de $D(X_k)$ tel que l'affectation partielle $\{(X_i,v),(X_k,w)\}$ satisfasse toutes les contraintes binaires sur X_i et X_k.
  - Le concept de consistance d'arc est « orienté »
    - La consistance de (X_i, X_k) n'implique pas la consistance de (X_k, X_i).
  - Un CSP est **arc-consistant** ssi tout les arcs (X_i, X_k) sont consistants.

Consistance d'Arc : CSP binaires

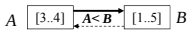
224

- Exemple

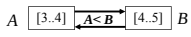
Aucun arc consistant



(A,B) consistant



(A,B) et (B,A) consistants



Consistance d'Arc : CSP quelconques

225

Arc Consistency :

- Soit $C_z(X_1, \ldots, X_n)$ une contrainte sur n variables
- C_z est arc consistante ssi, $\forall X_i$, pour chaque valeur $v \in X_i$ il existe des valeur $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots v_n$ dans $D(X_1),\ldots, D(X_{i-1}), D(X_{i+1}), \ldots, D(X_n)$ telles que

  $Cz(v_1, \ldots, v_{i-1}, v, v_{i+1}\ldots, v_n)$ soit vérifiée

CSP arc consistant

226

- Un CSP est **arc consistant** si **toutes ses contraintes sont arc-consistantes**.

Consistance d'Arc : Arc-B-Consistance

227

Arc-B-Consistency : (Arc-Bound-Consistency)

- Soit $C_z(X_1, \ldots, X_n)$ une contrainte sur n variables telles que $D(X_i) = [LB(X_i)..UB(X_i)]$ pour chaque X_i.
- C_z est arc-B-consistante ssi $\forall X_i \in X$, pour les deux valeurs $LB(X_i)$ et $UB(X_i)$, il existe des valeur $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots v_n$ et des valeur $w_1, \ldots, w_{i-1}, w_{i+1}, \ldots w_n$ dans $D(X_1),\ldots, D(X_{i-1}), D(X_{i+1}), \ldots D(X_n)$ telles que
  - $C_z(v_1, \ldots, v_{i-1}, LB(X_i), v_{i+1}\ldots, v_n)$ et
  - $C_z(w_1, \ldots, w_{i-1}, UB(X_i), w_{i+1}\ldots, w_n)$ soient vérifiées

CSP "consistant de chemin"

228

Path Consistency : un CSP (X, D, C), **consistant de chemin** si pour tout couple de variables (X_i, X_k) de X reliées par un chemin P dans le graphe de contraintes, il existe une affectation pour chaque variable le long de P qui satisfait toutes les contraintes binaires.

Faire respecter les contraintes

229

- Algorithmes qui enlèvent les valeurs des domaines des variables d'un CSP jusqu'à ce qu'il soit consistant de noeud, consistant d'arc ou consistant de chemin : **NC** (pour Node Consistency), **AC** (pour Arc Consistency), **PC** (pour Path Consistency).
  - on dit que les algorithmes **filtrent** les domaines des variables
  - Il existe différentes versions de ces algorithmes : AC1, AC2, AC3, ..., chaque version étant plus efficace (plus rapide) que la précédente.

Filtrage : Utilisation de AC-X et PC-X

230

- En utilisant les algorithmes de filtrage lors de l'exploration de l'espace de recherche, on enlève de nombreuses valeurs incompatibles des domaines.
- Si un domaine est vide, cela prouve qu'il n'existe pas de solution.
- Si tous les domaines sont réduits à un singleton : on a une solution.
- On n'obtient pas toujours une solution.
- On ne prouve pas toujours qu'il n'existe pas de solution.
- En général réduit les domaines et donc l'espace de recherche.

Filtrage : Utilisation de AC-X et PC-X

231

- AC-X et PC-X sont des algorithmes incomplets : il peut rester des valeurs dans les domaine de variable qui sont inconsistantes.

Filtrage et localité

232

- La modification du domaine d'une variable entraîne la modification du domaine de ses voisins, des voisins de ses voisins, … ⇒ propagation à travers le réseau de contraintes
- Parcours incontrôlé dans sa profondeur
- Plus il est important, plus la résolution du problème est rapide
- Processus qui peut être convergent (cyles du réseau de contraintes)
- Propagation = traitement local dynamique : réalisée automatiquement et librement par la dynamique interne des entités du réseau
- L'entité « contrainte » possède :
  - L'intelligence pour réduire les domaines des variables qu'elle relie

Propagation de contraintes dans l'énumération des solutions

233

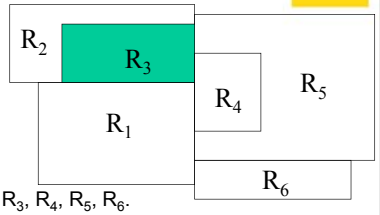
- exclure les valeurs incohérentes (inconsistantes) des domaines des variables en utilisant les contraintes lors de la recherche de solution (*e.g.*, avec `RetourArrière`).
- Ceci permet de réduire l'espace de recherche en élaguant l'arbre de recherche d'affectations reconnues inconsistantes.
- Les inconsistances ne seront plus seulement détectées lors du test de consistance d'une affectation partielle : les contraintes sont utilisées dès que possible (dès q'une décision est prise) pour réduire en amont les domaines de variables :

=> Propagation des contraintes

Coloriage de Cartes

234

- Problème : colorier une carte planaire de façon qu'aucune paire de de régions adjacentes ne soit de la même couleur.



Variables : R_1, R_2, R_3, R_4, R_5, R_6.

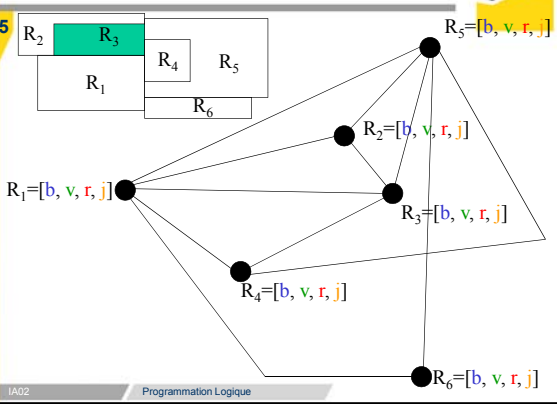
Domaines : $D(R_i)$=[rouge, vert, bleu, jaune].

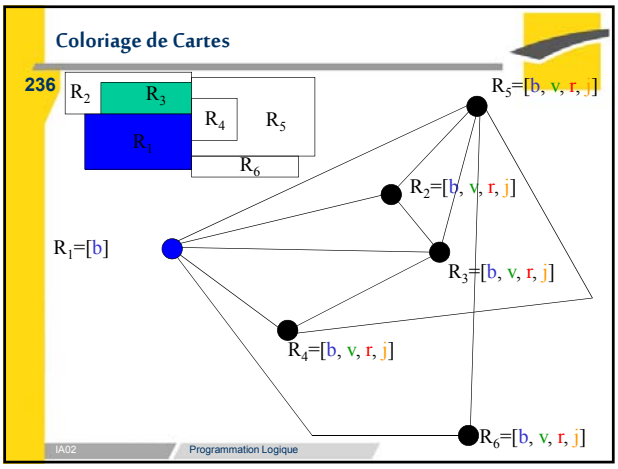
Contraintes :

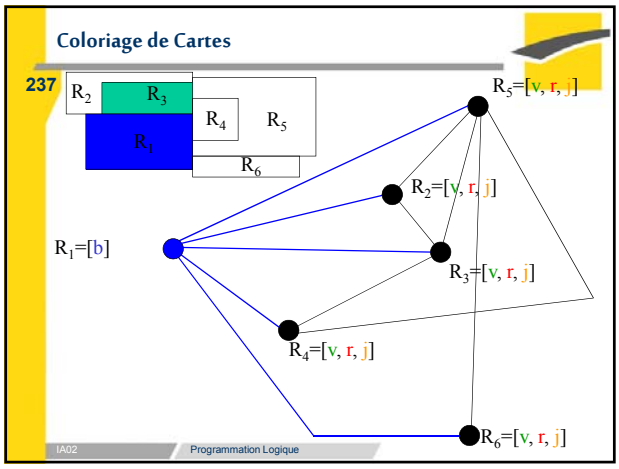
$R_1 \neq R_2$, $R_1 \neq R_3$, $R_1 \neq R_4$, $R_1 \neq R_5$, $R_1 \neq R_6$, $R_2 \neq R_3$, $R_2 \neq R_5$, $R_3 \neq R_4$, $R_3 \neq R_5$, $R_4 \neq R_5$, $R_5 \neq R_6$.

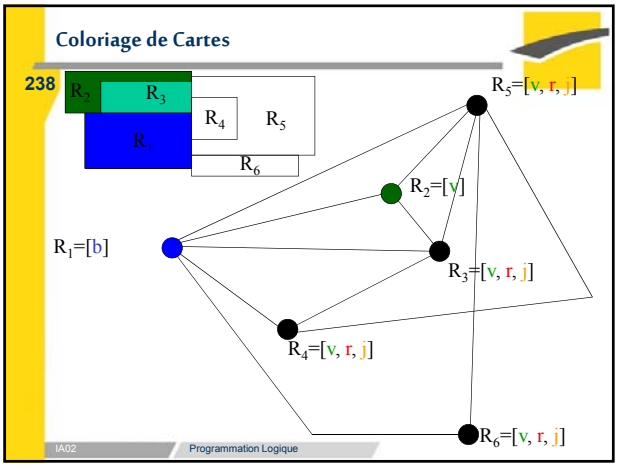
Coloriage de Cartes

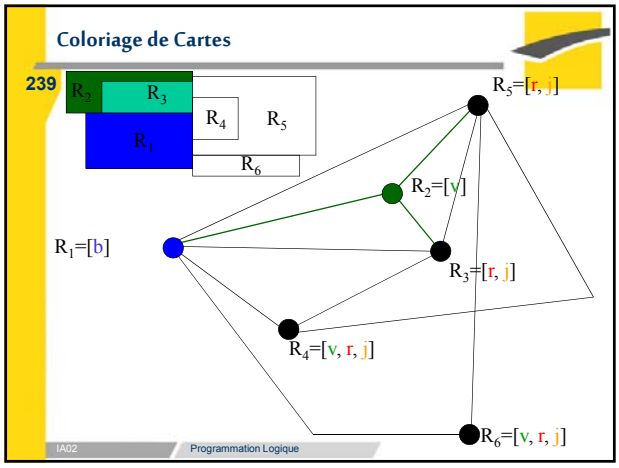
235

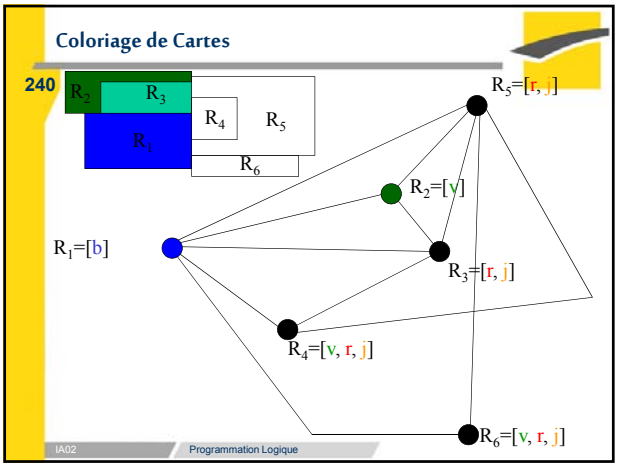


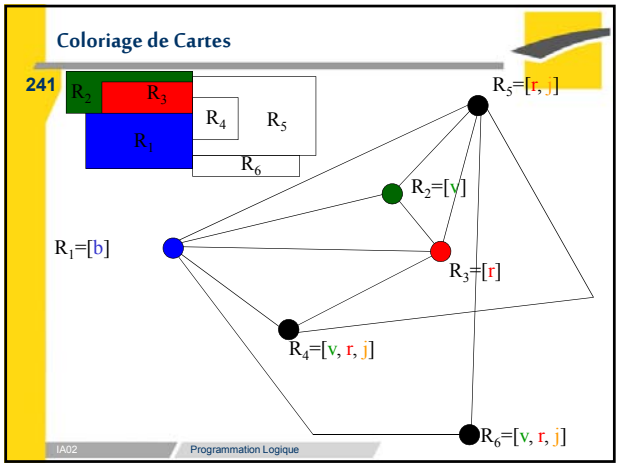

Coloriage de Cartes
236
R2
R3
R4
R5
R1
R6
R5=[b, v, r, j]
R2=[b, v, r, j]
R1=[b]
R3=[b, v, r, j]
R4=[b, v, r, j]
R6=[b, v, r, j]
IA02
Programmation Logique


Coloriage de Cartes
237
R2
R3
R4
R5
R1
R6
R5=[v, r, j]
R2=[v, r, j]
R1=[b]
R3=[v, r, j]
R4=[v, r, j]
R6=[v, r, j]
IA02
Programmation Logique


Coloriage de Cartes
238
R2
R3
R4
R5
R1
R6
R5=[v, r, j]
R2=[v]
R1=[b]
R3=[v, r, j]
R4=[v, r, j]
R6=[v, r, j]
IA02
Programmation Logique

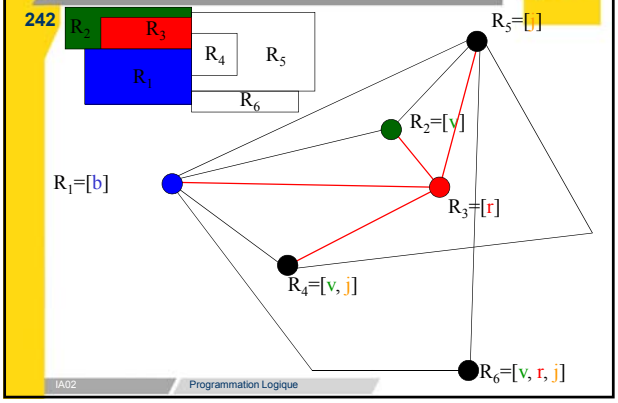

Coloriage de Cartes
239
R2
R3
R4
R5
R1
R6
R5=[r, j]
R2=[v]
R1=[b]
R3=[r, j]
R4=[v, r, j]
R6=[v, r, j]
IA02
Programmation Logique


Coloriage de Cartes
240
R2
R3
R4
R5
R1
R6
R5=[r, j]
R2=[v]
R1=[b]
R3=[r, j]
R4=[v, r, j]
R6=[v, r, j]
IA02
Programmation Logique


Coloriage de Cartes
241
R2
R3
R4
R5
R1
R6
R5=[r, j]
R2=[v]
R1=[b]
R3=[r]
R4=[v, r, j]
R6=[v, r, j]
IA02
Programmation Logique

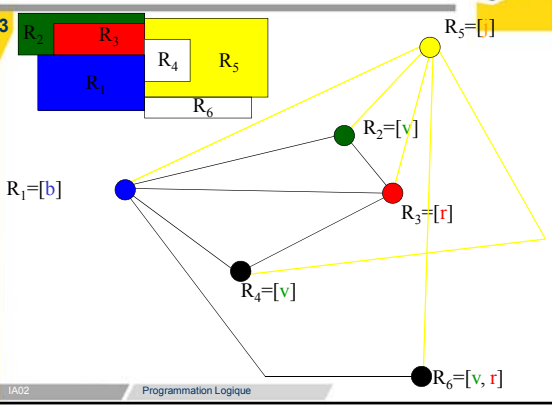
Coloriage de Cartes

242



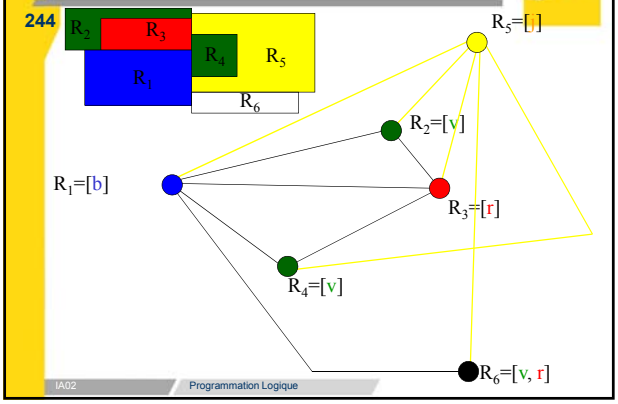
Coloriage de Cartes

243



Coloriage de Cartes

244



Intégrer des heuristiques

245

- Ces deux ordres peuvent changer considérablement l'efficacité de ces algorithmes :
  - Si on disposait d'un "oracle" qui nous disait quelle valeur choisir sans se tromper, la solution serait trouvée sans Backtrack.
    - Malheureusement, le problème général de la résolution d'un CSP sur les domaines finis étant NP-complet, il est plus qu'improbable que cet oracle fiable à 100% puisse jamais être "programmé".
  - On peut intégrer des heuristiques pour déterminer l'ordre dans lequel les variables et les valeurs doivent être considérées.
    - Une heuristique est une règle non systématique (elle n'est pas fiable) qui nous donne des indications sur la direction à prendre dans l'arbre de recherche.

Intégrer des heuristiques

246

- Les heuristiques concernant l'ordre d'instanciation des valeurs sont généralement dépendantes de l'application considérée et difficilement généralisables.
- En revanche, il existe de nombreuses heuristiques d'ordre d'instanciation des variables qui permettent bien souvent d'accélérer considérablement la recherche.
- L'idée générale consiste à **instancier en premier les variables les plus "critiques",**
  - c'est-à-dire celles qui interviennent dans beaucoup de contraintes et/ou celles qui ne peuvent prendre que très peu de valeurs.

Intégration de nouveaux outils en PPC : extensions

247

- Beaucoup d'algorithmes de propagation et de recherches sont intégrés dans les outils de programmation par contraintes.
- L'un des intérêts des outils de programmation par contraintes est qu'ils sont extensibles :
  - création de nouveaux objets
  - création de nouvelles contraintes par rapport à ces objets
  - création de nouvelles méthodes de propagation de ces contraintes
- Bibliothèques pré implémentées dans certains domaines (Ordonnancement et en Plannification : ILOG Scheduler)

Limites de la Programmation par Contraintes

248

- La programmation par contraintes ne fait pas de miracle :
  Les problèmes NP-durs restent NP-durs...
- La programmation par contraintes n'est pas une solution immédiate :
  - de nombreux outils sont disponibles
  - développer et tester les logiciels reste à faire...
- La programmation par contraintes n'est pas toujours **LA** solution :
  - de nombreux problèmes ne nécessitent pas l'utilisation de la programmation par contrainte (*e.g.*, recherche de chemin critique dans un graphe) ou sont tels que la programmation par contraintes n'apportent aucun bénéfice à leur résolution.