Programmation Par Contraintes

Aide à la Décision et Intelligence Artificielle

Licence 3

Dr. **Ouali Abdelkader**
abdelkader.ouali@unicaen.fr

Département Informatique
UFR des sciences
Université de Caen Normandie

2021

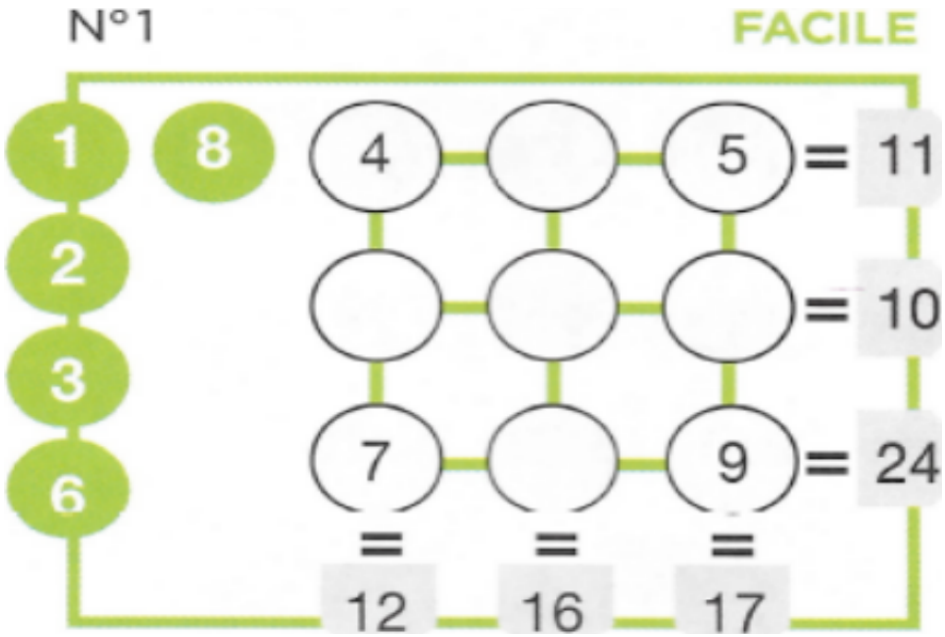
Sommaire

Contraintes et problèmes de satisfaction de contraintes :

Fubuki

- Étant donné un **ensemble de pions** chacun portant un chiffre (à gauche), et une **grille partiellement remplie** de chiffres (à droite)

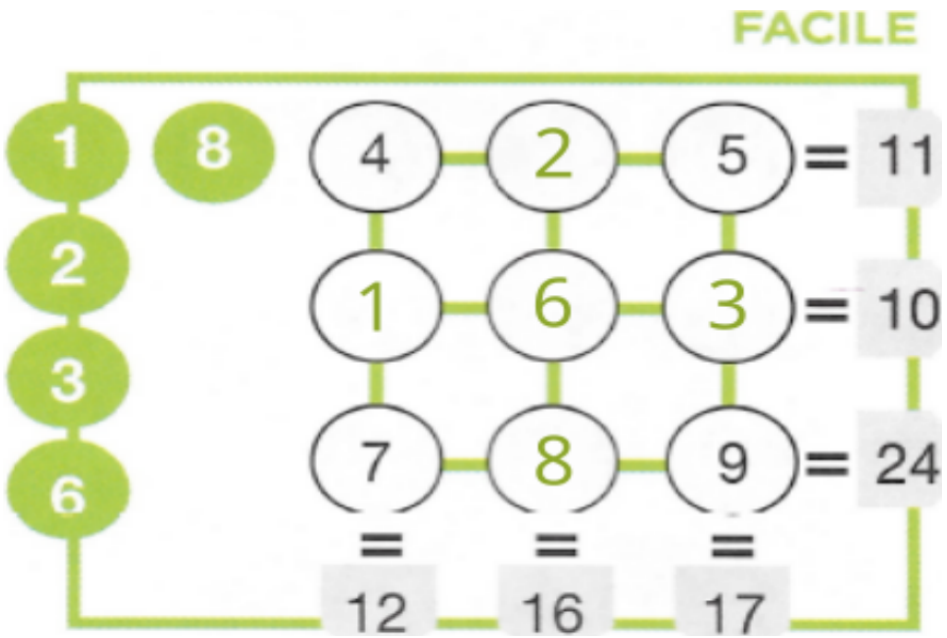
- **Objectif : disposer les pions dans la grille** afin que **la somme indiquée** pour chaque **ligne** et chaque **colonne** soit **satisfaite**



Fubuki

- Étant donné un **ensemble de pions** chacun portant un chiffre (à gauche), et une **grille partiellement remplie** de chiffres (à droite)

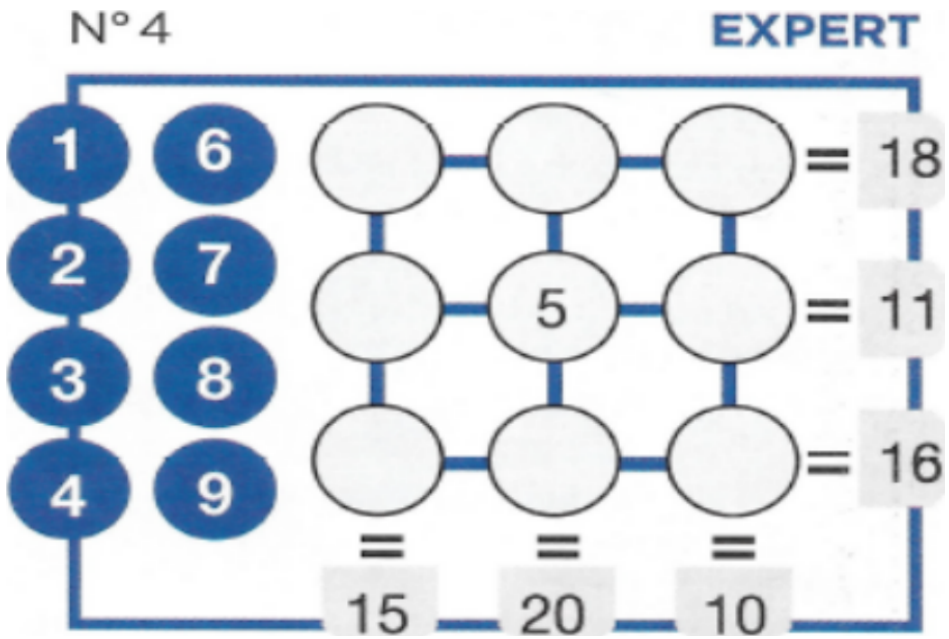
- **Objectif : disposer les pions dans la grille** afin que **la somme indiquée** pour chaque **ligne** et chaque **colonne** soit **satisfaite**



Fubuki

- Étant donné un **ensemble de pions** chacun portant un chiffre (à gauche), et une **grille partiellement remplie** de chiffres (à droite)

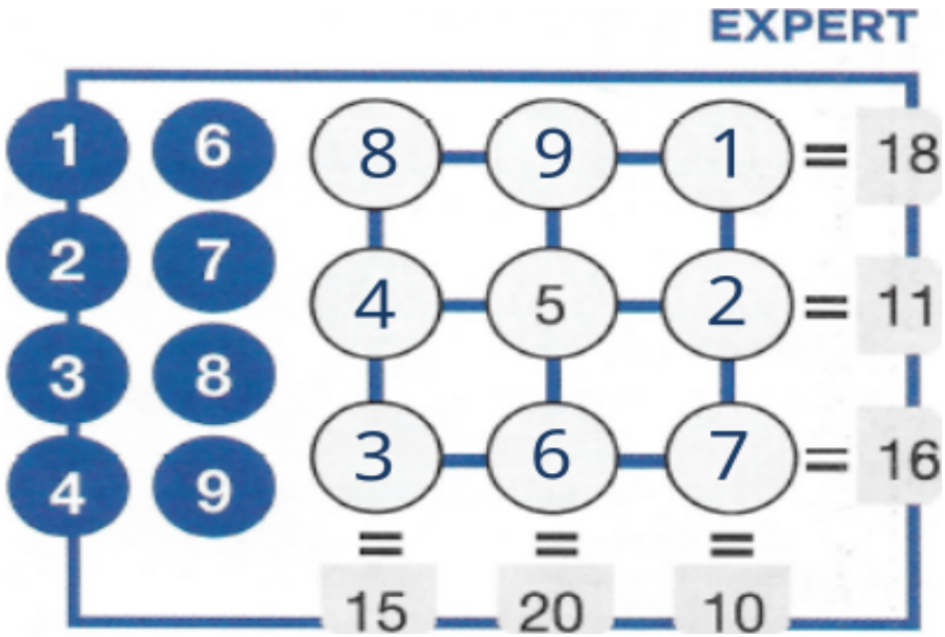
- **Objectif : disposer les pions dans la grille** afin que **la somme indiquée** pour chaque **ligne** et chaque **colonne** soit **satisfaite**



Fubuki

- Étant donné un **ensemble de pions** chacun portant un chiffre (à gauche), et une **grille partiellement remplie** de chiffres (à droite)

- **Objectif : disposer les pions dans la grille** afin que **la somme indiquée** pour chaque **ligne** et chaque **colonne** soit **satisfaite**



Intérêt

En général, nous avons de **nombreux problèmes** (e.g. affectation, planification, sélection, etc.) dans **différents domaines** (location, transport, recherche médicale, etc.) dont on :

- doit satisfaire un **ensemble de contraintes**
- avec des **approches de résolution** souvent **communes**

Intérêt

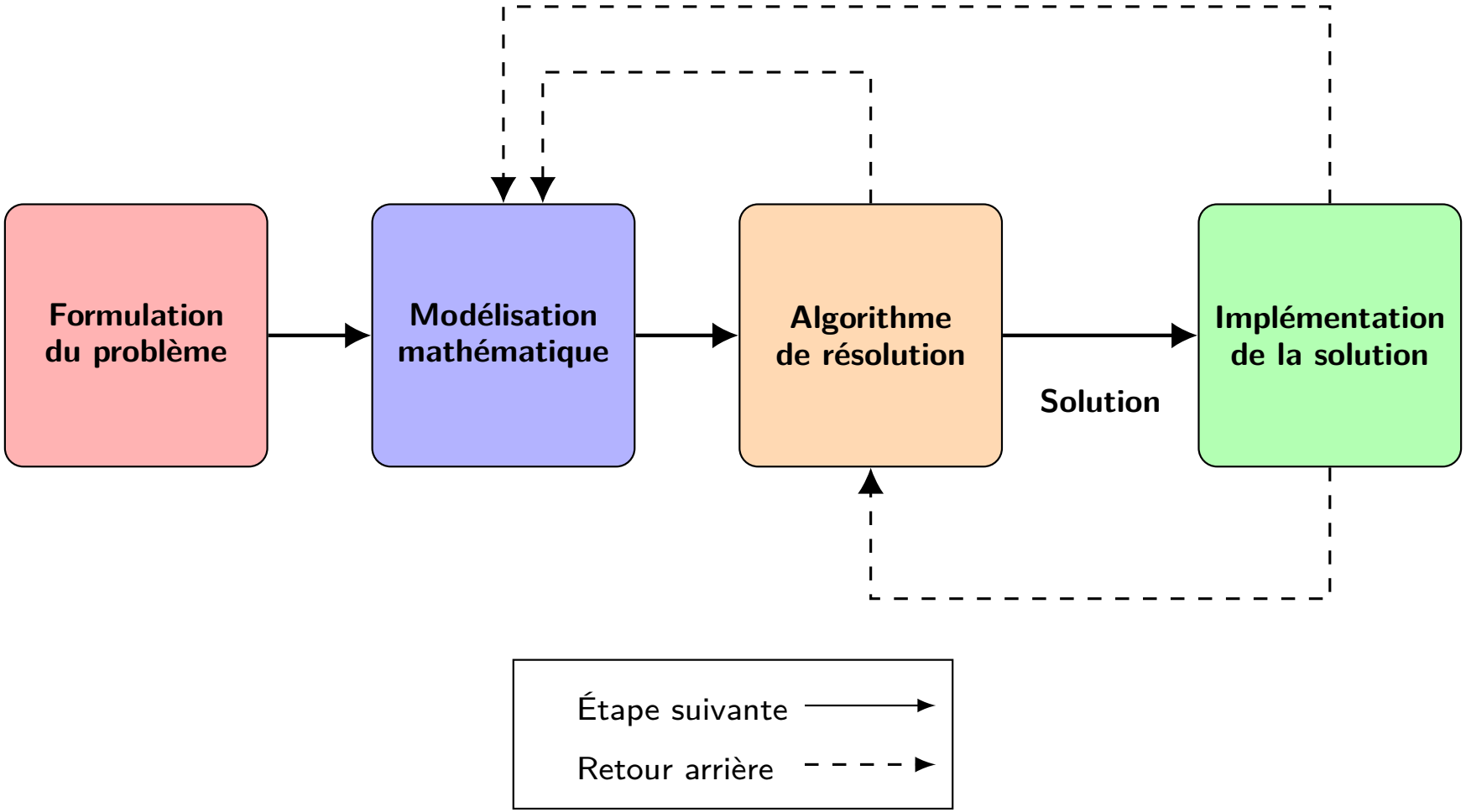
En général, nous avons de **nombreux problèmes** (e.g. affectation, planification, sélection, etc.) dans **différents domaines** (location, transport, recherche médicale, etc.) dont on :

- doit satisfaire un **ensemble de contraintes**
- avec des **approches de résolution** souvent **communes**

Motivation :

- Cadre **flexible**
  ➥ on peut résoudre des puzzles, de l'ordonnancement, du partitionnement, de la coloration de graphe, etc.
- Résolution **automatique** et **générique**
  ➥ sans écrire un algorithme spécifique pour chaque problème

Processus de prise de décision



Contraintes

- Une contrainte est une **relation** définie sur un nombre **fini de variables**
  ➥ exemple d'une contrainte sur **deux variables** (**inconnues**) :
  $(x_1 = 1 \leftrightarrow x_2 = 1)$, $(x_1 = 2 \leftrightarrow x_2 = 2)$, $(x_1 = 3 \leftrightarrow x_2 = 3)$

Contraintes

- Une contrainte est une **relation** définie sur un nombre **fini de variables**
  ➥ exemple d'une contrainte sur **deux variables** (**inconnues**) :
  $(x_1 = 1 \leftrightarrow x_2 = 1)$, $(x_1 = 2 \leftrightarrow x_2 = 2)$, $(x_1 = 3 \leftrightarrow x_2 = 3)$

- Une variable admet un **domaine d'instanciation** de valeurs
  ➥ exemple d'un **domaine** d'une variable :
  $x_1 = x_2 = \{1, 2, 3\}$

Contraintes

- Une contrainte est une **relation** définie sur un nombre **fini de variables**
  ➥ exemple d'une contrainte sur **deux variables** (**inconnues**) :
  $(x_1 = 1 \leftrightarrow x_2 = 1), (x_1 = 2 \leftrightarrow x_2 = 2), (x_1 = 3 \leftrightarrow x_2 = 3)$

- Une variable admet un **domaine d'instanciation** de valeurs
  ➥ exemple d'un **domaine** d'une variable :
  $x_1 = x_2 = \{1, 2, 3\}$

- Une contrainte **restreint** les valeurs que peuvent prendre simultanément les variables qu'elle relie :
  ➥ **instanciations interdites** par la contrainte :
  $\cancel{(x_1 = 1 \leftrightarrow x_2 = 2)}, \cancel{(x_1 = 1 \leftrightarrow x_2 = 3)}, \cancel{(x_1 = 2 \leftrightarrow x_2 = 1)},$
  $\cancel{(x_1 = 2 \leftrightarrow x_2 = 3)}, \cancel{(x_1 = 3 \leftrightarrow x_2 = 1)}, \cancel{(x_1 = 3 \leftrightarrow x_2 = 2)}$

Problème de Satisfaction de Contraintes

Un **CSP** (Constraint Satisfaction Problem) est défini par un triplet $P = (X, D, C)$:

Problème de Satisfaction de Contraintes

Un **CSP** (Constraint Satisfaction Problem) est défini par un triplet $P = (X, D, C)$:

CSP

- $X = \{x_1, \ldots, x_n\}$ un **ensemble fini de n variables**.
- D une **fonction** associant à chaque variable x_i un **domaine fini de valeurs** $D(x_i)$, $1 \leq i \leq n$.
- $C = \{C_1, \ldots, C_m\}$ un **ensemble fini de m contraintes** sur les variables du problème P.

Définition d'une contrainte

Définition en **extension** : toutes les valeurs autorisées des variables dans la portée de la contrainte sont **explicitement** spécifiées :

- $(x_1 = 1 \leftrightarrow x_2 = 1)$
- $(x_1 = 2 \leftrightarrow x_2 = 2)$
- $(x_1 = 3 \leftrightarrow x_2 = 3)$

Définition d'une contrainte

Définition en **extension** : toutes les valeurs autorisées des variables dans la portée de la contrainte sont **explicitement** spécifiées :

- $(x_1 = 1 \leftrightarrow x_2 = 1)$
- $(x_1 = 2 \leftrightarrow x_2 = 2)$
- $(x_1 = 3 \leftrightarrow x_2 = 3)$

Définition en **compréhension** : la relation entre les variables impliquées est exprimée en fonction **d'opérateurs arithmétiques** quand c'est possible, ou avec un **nom** quand la contrainte est complexe :

- $x_1 = x_2$
- $\mathit{égalité}(x_1, x_2)$

Instanciation et Solution d'un CSP

- Une **instanciation** associe à **chacune** des variables du CSP **un élément** de son **domaine**
  ➠ c'est un élément du **produit cartésien** : $D(x_1) \times ... \times D(x_n)$

$$\underbrace{\{1,2,3\}}_{x1} \times \underbrace{\{1,2,3\}}_{x2} = \{\mathbf{(1,1)},\ (1,2),\ (1,3),\ (2,1),\ \ldots,\ (3,3)\}$$

Instanciation et Solution d'un CSP

- Une **instanciation** associe à **chacune** des variables du CSP **un élément** de son **domaine**

  ➠ c'est un élément du **produit cartésien** : $D(x_1) \times ... \times D(x_n)$

$$\underbrace{\{1,2,3\}}_{\text{x1}} \times \underbrace{\{1,2,3\}}_{\text{x2}} = \{\mathbf{(1,1)},\ (1,2),\ (1,3),\ (2,1),\ \ldots,\ (3,3)\}$$

- Une **solution** est une instanciation **satisfaisant chacune des contraintes** du CSP

  ➠ Si dans **chaque contrainte** on remplace **chacune des variables** dans sa portée par la **valeur** que lui affecte l'**instanciation**, alors ladite contrainte s'évalue à **vrai**

Cryptarithmétique[1] : HOMME + FEMME = PARITE

Description du problème

Dans cette addition cryptée sur différentes lettres, on considère que :

- à chaque **lettre** est associé un **chiffre**, et
- **aucune** lettre n'a le **même** chiffre d'une **autre** lettre.

Il existe **plusieurs solutions** qui, si parité oblige, permettent d'**intervertir le chiffre** de **H** et celui de **F**.

Question

Quelle est la valeur du mot **PARITE** qui vérifie la contrainte d'addition, sachant que si on construit le mot **PAIR** avec lettres, il résulte à un nombre **pair** ?

$$\begin{array}{cccccccc} & & H & O & M & M & E \\ + & & F & E & M & M & E \\ \hline = & P & A & R & I & T & E \end{array}$$

1. Élisabeth Busser, et al. www.tangente-mag.com

CSP : HOMME + FEMME = PARITE

$$\begin{array}{cccccccc} & & H & O & M & M & E \\ + & & F & E & M & M & E \\ \hline = & P & A & R & I & T & E \end{array}$$

Variables :

CSP : HOMME + FEMME = PARITE

$$\begin{array}{cccccccc} & & H & O & M & M & E \\ + & & F & E & M & M & E \\ \hline = & P & A & R & I & T & E \end{array}$$

Variables :

- A, E, F, H, I, M, O, P, R et T

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Variables :

- A, E, F, H, I, M, O, P, R et T
- c_1, c_2, c_3 et c_4

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Variables :

- A, E, F, H, I, M, O, P, R et T
- c_1, c_2, c_3 et c_4

Domaines :

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Variables :

- A, E, F, H, I, M, O, P, R et T
- c_1, c_2, c_3 et c_4

Domaines :

- $D(A) = D(E), \ldots, D(T) = \{0, 1, \ldots, 9\}$

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Variables :

- A, E, F, H, I, M, O, P, R et T
- c_1, c_2, c_3 et c_4

Domaines :

- $D(A) = D(E), \ldots, D(T) = \{0, 1, \ldots, 9\}$
- $D(c_1) = D(c_2) = D(c_3) = D(c_4) = \{0, 1\}$

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Contraintes :

- $E + E = E + 10\ c_1$
- $c_1 + M + M = T + 10\ c_2$
- $c_2 + M + M = I + 10\ c_3$
- $c_3 + O + E = R + 10\ c_4$
- $c_4 + H + F = A + 10\ P$

CSP : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	E
+		F	E	M	M	E
=	P	A	R	I	T	E

Contraintes :

- $E + E = E + 10\ c_1$
- $c_1 + M + M = T + 10\ c_2$
- $c_2 + M + M = I + 10\ c_3$
- $c_3 + O + E = R + 10\ c_4$
- $c_4 + H + F = A + 10\ P$
- $alldifferent(A, E, F, H, I, M, O, P, R \text{ et } T)$

Solution : HOMME + FEMME = PARITE

Le **solveur** nous a retourné **deux solutions** :

		0	1	1	0	
		9	7	6	6	0
+		**5**	0	6	6	0
=	1	4	8	3	2	0

		0	1	1	0	
		5	7	6	6	0
+		**9**	0	6	6	0
=	1	4	8	3	2	0

PAIR = **1438**, qui est un nombre pair

- Chaque valeur trouvée **appartient** au domaine de la variable
- Toutes les contraintes sont **satisfaites** par les deux solutions

Observations : HOMME + FEMME = PARITE

		c_4	c_3	c_2	c_1	
		H	O	M	M	**E**
+		F	**E**	M	M	**E**
=	**P**	A	R	**I**	T	**E**

- Aucun chiffre doublé autre que 0 ne redonne la même unité
- Les unités ne diffèrent que d'une unité quand on a deux additions successives de la même lettre
- L'addition d'un chiffre avec 0 n'engendre pas de retenue

Modèle CSP amélioré

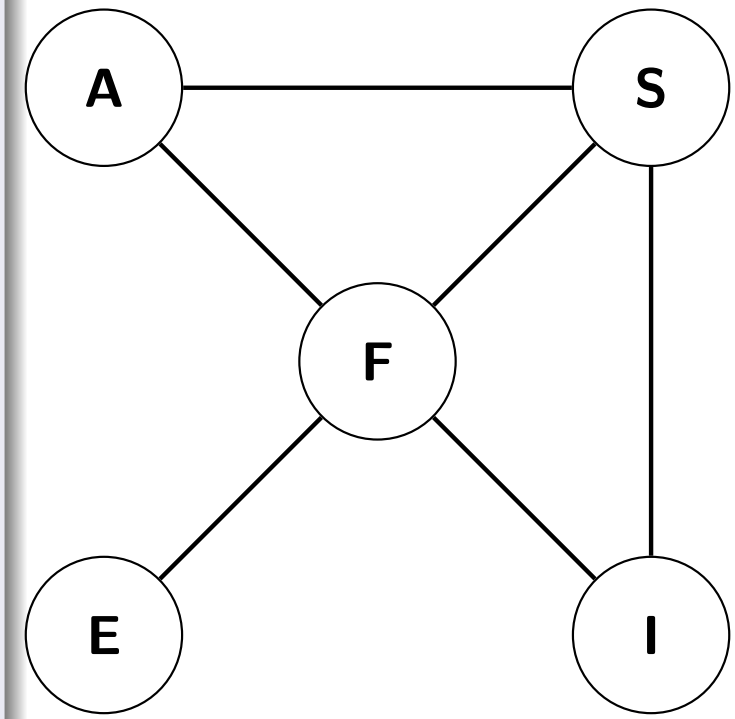
Écrire un modèle **CSP équivalent** au CSP précédent mais qui est **plus réduit** en considérant les **différentes observations**.

Exemple : le problème de coloriage d'un graphe

Description

On dispose d'un **graphe non orienté** qui est une représentation graphique d'une carte géographique à colorier avec une **fonction cl** associant à chaque **nœud** du graphe un **ensemble de couleurs permises**, où :

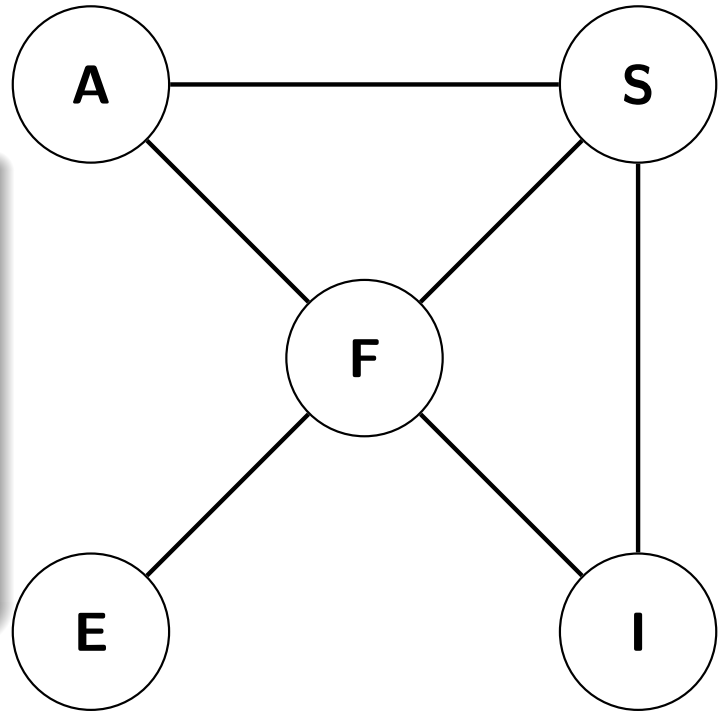
- Les **nœuds** du graphe sont les différents **pays** de la carte
- Les **couleurs** associées à un nœud sont celles avec lesquelles le pays correspondant peut être colorié
- Deux nœuds sont **adjacents** si et seulement si les pays correspondants ont une **frontière terrestre** commune



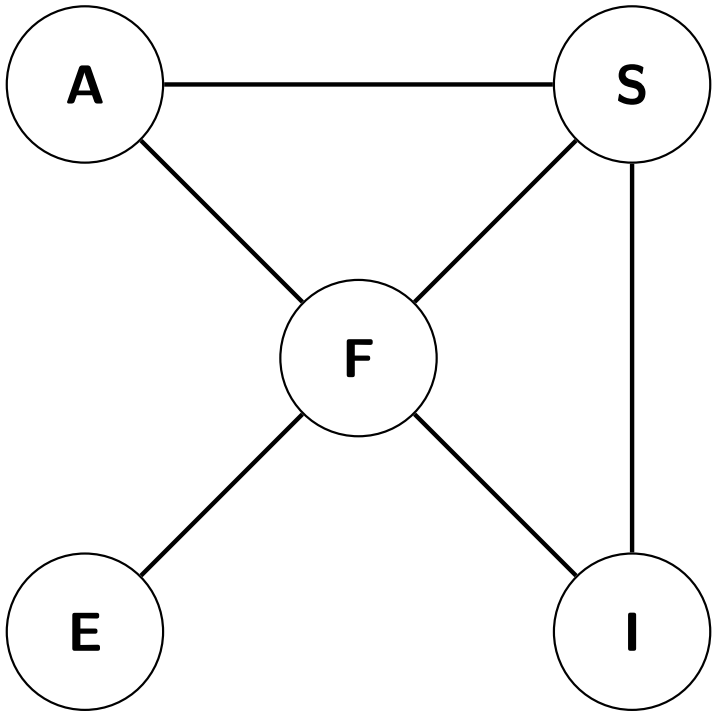
Exemple : le problème de coloriage d'un graphe

Question

Trouver le modèle CSP pour **colorier** les nœuds du graphe, **chacun** avec **une couleur** de l'ensemble que lui associe la **fonction cl**, de telle sorte que **deux noeuds adjacents** (i.e., reliés par une arête) **n'aient pas la même couleur** ?

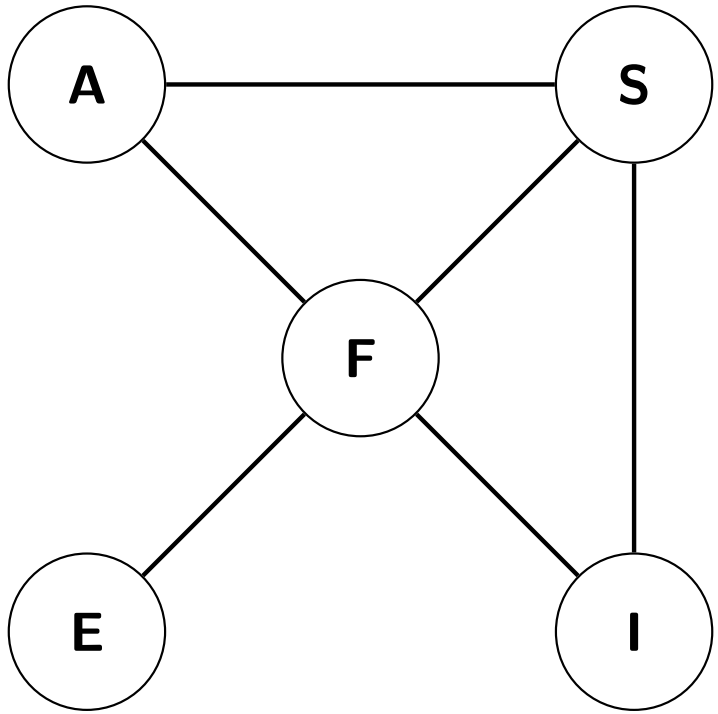


Exemple : le problème de coloriage d'un graphe



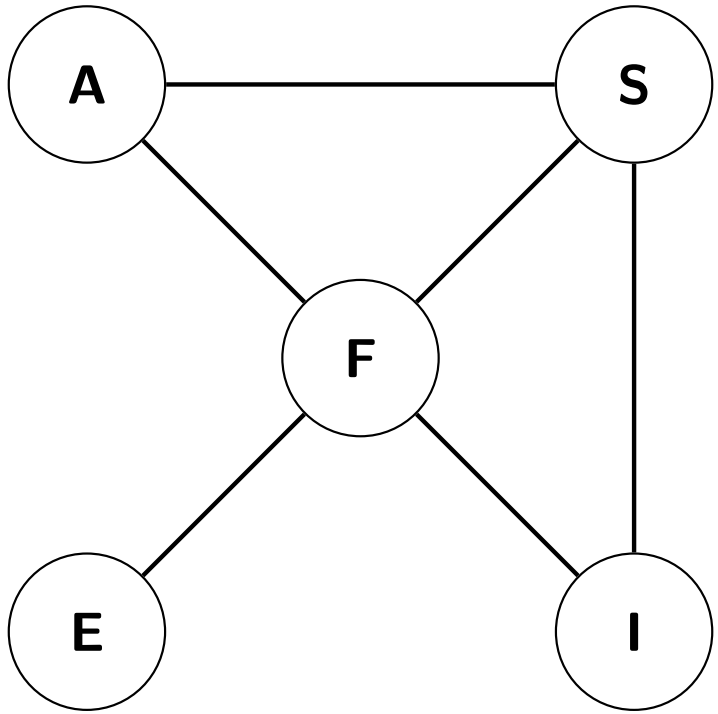
Exemple : le problème de coloriage d'un graphe

- **Variables :**
  - A, S, F, I et E
- **Domaines :**
  - $\{$**rouge**, **vert**, **bleu**$\}$



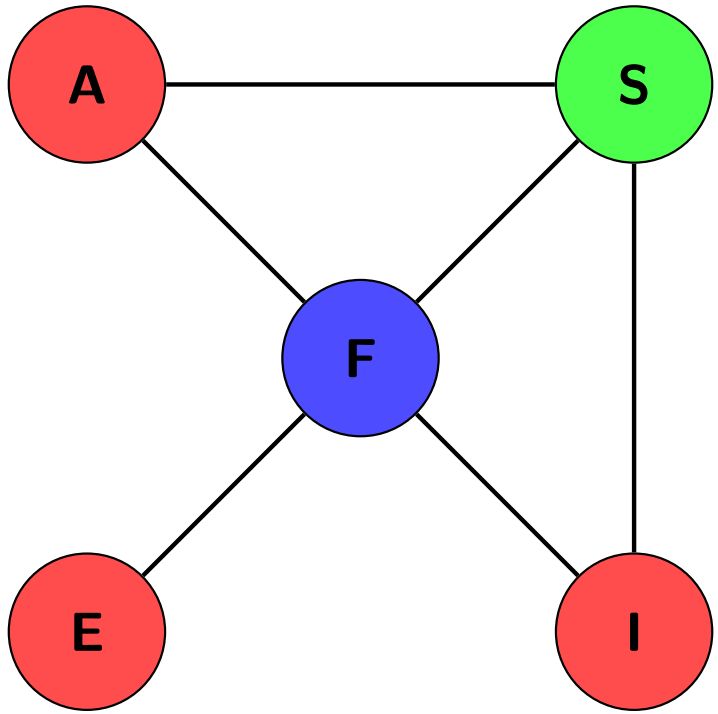
Exemple : le problème de coloriage d'un graphe

- **Variables :**
  - A, S, F, I et E
- **Domaines :**
  - $\{$**rouge**, **vert**, **bleu**$\}$
- **Contraintes :**
  - $A \neq S, A \neq F, S \neq F,$
  - $S \neq I, F \neq I$ et $F \neq E$



Solution : le problème de coloriage d'un graphe

- Le **solveur** nous a retourné la **solution suivante** :
  - A = E = I = **rouge**,
  - S = **vert**, et
  - F = **bleu**



Prochaine séance

1. Types des CSP
2. Représentation graphique
3. Résolution

Programmation Par Contraintes

Aide à la Décision et Intelligence Artificielle

Licence 3

Dr. **Ouali Abdelkader**
abdelkader.ouali@unicaen.fr

Département Informatique
UFR des sciences
Université de Caen Normandie

2021

Sommaire

Types de CSP :

1. CSP binaire discret et continu
2. Représentation graphique
3. Résolution d'un CSP

Types de CSP

Portée d'une contrainte (**scope** en anglais) est l'ensemble des **variables impliquées** dans la **contrainte**.
On définit les variables $\{case_1, case_2, case_3\}$ et on considère les contraintes suivantes :

- $case_1 = 3$, **sa portée** : $\{case_1\}$
- $case_2 + case_3 = 12$, **sa portée** : $\{case_2, case_3\}$

Types de CSP

Portée d'une contrainte (**scope** en anglais) est l'ensemble des **variables impliquées** dans la **contrainte**.
On définit les variables $\{case_1, case_2, case_3\}$ et on considère les contraintes suivantes :

- $case_1 = 3$, **sa portée** : $\{case_1\}$
- $case_2 + case_3 = 12$, **sa portée** : $\{case_2, case_3\}$

CSP **binaire**

Chacune des contraintes porte sur **au plus deux variables**, si on prend l'ensemble de variables $\{x_1, x_2, x_3, \ldots, x_n\}$:

- $x_1 = 3$ (unaire)
- $x_1 = x_2$ (binaire)

Exemple de contraintes qui **ne doivent pas apparaître** dans **ce type** de CSP :

- ~~$x_1 + x_3 = x_2$~~ (ternaire)
- ~~$alldifferent(x_1, \ldots, x_n)$~~ (n-aires)

Types de CSP

CSP **discret**

Toutes les variables du CSP ont un **domaine fini**

- Un sous-ensemble **fini** de **nombres pairs** :
  $D_{10}(Pairs) = \{0, 2, 4, 6, 8, 10\}$
- Un ensemble fini de **couleurs** :
  $D(Couleurs) = \{Blue, Vert, Blanc\}$
- Un ensemble de **personnes** :
  $D(\text{Étudiant}) = \{Ania, Inès, Yun\}$

Types de CSP

CSP **continu**

Le domaine de **chacune** des variables est **continu**

- Un ensemble **continu** des points du temps, noté T :
- Un ensemble **continu** des points du plan, noté T^2 :
  intervalles de temps où toutes les paires (d,f) de T^2 vérifiant $d < f$

Exemple : temps de **début** et de **fin** des observations du **télescope Hubble**

CSP binaire discret

$P = (X, D, C)$

- $X = \{x_1, x_2, \ldots, x_n\}$
- $D = \{D(x_1), D(x_2), \ldots, D(x_n)\}$: **tous** les domaines sont **discrets** et **finis**
- $C = \{c_1, c_2 \ldots, c_m\}$: **toutes** les contraintes sont **unaires** ou **binaires**

Exemple :

- $X = \{x_1, x_2\}$
- $D(x_1) = D(x_2) = \{1, 2, \ldots, 100\}$
- $C = \{1 \leq x_1 \leq 9, \quad x_1^2 = x_2\}$

Représentation matricielle d'une contrainte

- On considère les **relations** $R_k(x_i, x_j)$ et $R_k(x_j, x_i)$ associées à la **contrainte** c_k d'un CSP **binaire discret** P, où $D(x_i) = \{a_1, a_2, \ldots, a_{|D(x_i)|}\}$ et $D(x_j) = \{b_1, b_2, \ldots, b_{|D(x_j)|}\}$

Représentation matricielle d'une contrainte

- On considère les **relations** $R_k(x_i, x_j)$ et $R_k(x_j, x_i)$ associées à la **contrainte** c_k d'un CSP **binaire discret** P, où $D(x_i) = \{a_1, a_2, \ldots, a_{|D(x_i)|}\}$ et $D(x_j) = \{b_1, b_2, \ldots, b_{|D(x_j)|}\}$
  - ➥ $R_k(x_i, x_j) = \{(a, b) = D(x_i) \times D(x_j) : (x_i, x_j) = (a, b) \text{ satisfait } c_k\}$
    $R_k(x_j, x_i) = \{(a, b) = D(x_j) \times D(x_i) : (x_j, x_i) = (a, b) \text{ satisfait } c_k\}$

Représentation matricielle d'une contrainte

- On considère les **relations** $R_k(x_i, x_j)$ et $R_k(x_j, x_i)$ associées à la **contrainte** c_k d'un CSP **binaire discret** P, où $D(x_i) = \{a_1, a_2, \ldots, a_{|D(x_i)|}\}$ et $D(x_j) = \{b_1, b_2, \ldots, b_{|D(x_j)|}\}$
  ➥ $R_k(x_i, x_j) = \{(a, b) = D(x_i) \times D(x_j) : (x_i, x_j) = (a, b) \text{ satisfait } c_k\}$
  $R_k(x_j, x_i) = \{(a, b) = D(x_j) \times D(x_i) : (x_j, x_i) = (a, b) \text{ satisfait } c_k\}$

- La relation $R_k(x_i, x_j)$ peut être **représentée** par une **matrice booléenne** $M_k(x_i, x_j)$ d'une **dimension** $(|D(x_i)| \times |D(x_j)|)$

Représentation matricielle d'une contrainte

- On considère les **relations** $R_k(x_i, x_j)$ et $R_k(x_j, x_i)$ associées à la **contrainte** c_k d'un CSP **binaire discret** P, où $D(x_i) = \{a_1, a_2, \ldots, a_{|D(x_i)|}\}$ et $D(x_j) = \{b_1, b_2, \ldots, b_{|D(x_j)|}\}$
  ➥ $R_k(x_i, x_j) = \{(a, b) = D(x_i) \times D(x_j) : (x_i, x_j) = (a, b) \text{ satisfait } c_k\}$
  $R_k(x_j, x_i) = \{(a, b) = D(x_j) \times D(x_i) : (x_j, x_i) = (a, b) \text{ satisfait } c_k\}$

- La relation $R_k(x_i, x_j)$ peut être **représentée** par une **matrice booléenne** $M_k(x_i, x_j)$ d'une **dimension** $(|D(x_i)| \times |D(x_j)|)$

- La **valeur associée** à chaque **élément** de la matrice :

$$M_k(x_i, x_j)[p, q] = \begin{cases} 1 & \text{Si } (x_i, x_j) = (a_p, b_q) \text{ satisfait la contrainte } c_k \\ 0 & \text{Sinon} \end{cases}$$

Représentation matricielle d'une contrainte

- On considère les **relations** $R_k(x_i, x_j)$ et $R_k(x_j, x_i)$ associées à la **contrainte** c_k d'un CSP **binaire discret** P, où $D(x_i) = \{a_1, a_2, \ldots, a_{|D(x_i)|}\}$ et $D(x_j) = \{b_1, b_2, \ldots, b_{|D(x_j)|}\}$
  - ➥ $R_k(x_i, x_j) = \{(a, b) = D(x_i) \times D(x_j) : (x_i, x_j) = (a, b) \text{ satisfait } c_k\}$
    $R_k(x_j, x_i) = \{(a, b) = D(x_j) \times D(x_i) : (x_j, x_i) = (a, b) \text{ satisfait } c_k\}$

- La relation $R_k(x_i, x_j)$ peut être **représentée** par une **matrice booléenne** $M_k(x_i, x_j)$ d'une **dimension** $(|D(x_i)| \times |D(x_j)|)$

- La **valeur associée** à chaque **élément** de la matrice :

$$M_k(x_i, x_j)[p, q] = \begin{cases} 1 & \text{Si } (x_i, x_j) = (a_p, b_q) \text{ satisfait la contrainte } c_k \\ 0 & \text{Sinon} \end{cases}$$

➠ Même raisonnement pour la relation $R_k(x_j, x_i)$

Représentation matricielle d'une contrainte

Exemple :

- $X = \{x_1, x_2\}$
- $D(x_1) = D(x_2) = \{a, b, c\}$
- $C = \{x_1 = x_2\}$

La matrice associée à la relation $x_1 = x_2$:

		x_2		
		a	b	c
x_1	a	1	0	0
	b	0	1	0
	c	0	0	1

Intersection de deux matrices booléennes

- On considère deux matrices booléennes M et N de dimension $(m \times n)$ chacune

- L'intersection de M et N est la matrice booléenne de dimension $(m \times n)$ colonnes notée $I(M, N) = M \cap N$:

Pour tout i=1..m, pour tout j=1..n :

$$I(M,N)[i,j] = \begin{cases} 1 & \text{Si } M[i,j] = 1 \wedge N[i,j] = 1 \\ 0 & \text{Sinon} \end{cases}$$

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :
  - L'ensemble des **sommets** de G est l'ensemble des **variables** X de P

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :
  - L'ensemble des **sommets** de G est l'ensemble des **variables** X de P
  - Pour toute **contrainte binaire** $c_k(x_i, x_j)$, G contient **un seul arc** (x_i, x_j) ou (x_j, x_i), **sinon aucun**

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :
  - L'ensemble des **sommets** de G est l'ensemble des **variables** X de P
  - Pour toute **contrainte binaire** $c_k(x_i, x_j)$, G contient **un seul arc** (x_i, x_j) ou (x_j, x_i), **sinon aucun**
  - Pour tout **arc** (x_i, x_j) de G, l'étiquette de (x_i, x_j) est l'**intersection de toutes les matrices booléennes** $M_k(x_i, x_j)$ représentant les relations $R_k(x_i, x_j)$ associées aux **différentes contraintes** $c_k(x_i, x_j)$ de P

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :
  - L'ensemble des **sommets** de G est l'ensemble des **variables** X de P
  - Pour toute **contrainte binaire** $c_k(x_i, x_j)$, G contient **un seul arc** (x_i, x_j) ou (x_j, x_i), **sinon aucun**
  - Pour tout **arc** (x_i, x_j) de G, l'étiquette de (x_i, x_j) est l'**intersection de toutes les matrices booléennes** $M_k(x_i, x_j)$ représentant les relations $R_k(x_i, x_j)$ associées aux **différentes contraintes** $c_k(x_i, x_j)$ de P

Exemple

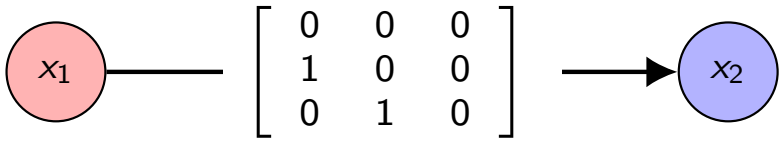
- $X = \{x_1, x_2\}$
- $D(x_1) = D(x_2) = \{1, 2, 3\}$
- $C = \{x_1 >= x_2,\ x_1 = x_2 + 1\}$

Représentation graphique d'un CSP binaire

- Un CSP binaire discret $P = (X, D, C)$ est un **graphe orienté étiqueté** $G = (X, E, L)$ défini comme suit :
  - L'ensemble des **sommets** de G est l'ensemble des **variables** X de P
  - Pour toute **contrainte binaire** $c_k(x_i, x_j)$, G contient **un seul arc** (x_i, x_j) ou (x_j, x_i), **sinon aucun**
  - Pour tout **arc** (x_i, x_j) de G, l'étiquette de (x_i, x_j) est l'**intersection de toutes les matrices booléennes** $M_k(x_i, x_j)$ représentant les relations $R_k(x_i, x_j)$ associées aux **différentes contraintes** $c_k(x_i, x_j)$ de P

Exemple

- $X = \{x_1, x_2\}$
- $D(x_1) = D(x_2) = \{1, 2, 3\}$
- $C = \{x_1 >= x_2,\ x_1 = x_2 + 1\}$



Résolution d'un CSP

Instanciation

- L'**instanciation** d'une variable est le couple **(variable, valeur)** où valeur est un des éléments du domaine de la variable
- Une instanciation est **partielle** s'il y a des variables du CSP ayant un domaine contenant **deux ou plusieurs valeurs**
- Une instanciation est **complète** si le domaine de chaque variable du CSP a **une valeur**

Instanciation

- L'**instanciation** d'une variable est le couple **(variable, valeur)** où valeur est un des éléments du domaine de la variable
- Une instanciation est **partielle** s'il y a des variables du CSP ayant un domaine contenant **deux ou plusieurs valeurs**
- Une instanciation est **complète** si le domaine de chaque variable du CSP a **une valeur**

Exemple : (variables = {couleurToit et couleurCapot})

- (couleurToit, "noir") incomplète
- (couleurCapot, "rouge") incomplète
- (couleurToit, "noir"), (couleurCapot, "rouge") complète

Vérification d'une contrainte

- Une contrainte **se vérifie** sur une **instanciation**
- Une contrainte est vérifiée (ou satisfaite) si les valeurs des variables **respectent** la contrainte

Vérification d'une contrainte

- Une contrainte **se vérifie** sur une **instanciation**
- Une contrainte est vérifiée (ou satisfaite) si les valeurs des variables **respectent** la contrainte

Exemple : (variables : A={1,2,3} et B={0,1,2,3}, contrainte : A + B = 2)

- {(A, 2)}, pas toute la portée assignée, **invérifiable**
- {(A, 2),(B, 2)}, toute la portée assignée, **vérifiable**
- 2+2=2 est faux, **non satisfaite**
- {(A, 2),(B, 0)}, toute la portée assignée, **vérifiable**
- 2+0=2 est vrai, **satisfaite**

Solution

- Une solution est une **instanciation complète** et **valide**

Exemples : (contrainte A + B = 2)

- {(A,2)}, **non complète** : **pas une solution**
- {(A,2),(B,2)}, **complète**, mais **invalide** : **pas une solution**
- {(A,2),(B,0)}, **complète** et **valide** : **une solution**

Architecture d'un solveur de PPC

1. Réduction du problème CSP

- Exécuter des algorithmes sur les contraintes du CSP **sans faire d'hypothèses**
- Obtenir un CSP **plus simple** (instanciation de variables, réduire les domaines) et cohérent
- Dans certaines situations, un **retour immédiatement** (backtracker) est possible : **pas de solution** au problème

  ➥ **arcs de cohérence :** calcul et réduction des domaines

Architecture d'un solveur de PPC

2. Explorations de l'espace de solutions avec une hypothèse

- Choix d'une **variable** $x_i \in X$ **non instanciée**
- Choix d'une **valeur** $k \in D(x_i)$ **non explorée**
- Utilisation des **heuristiques**

➥ Prendre en compte **la spécificité** du problème pour **trouver rapidement** des solutions

➥ Filtrage qui **réduit bien les domaines** des variables à partir des **choix causés initialement**

Architecture d'un solveur de PPC

3. Vérification de l'hypothèse

- Pour toute contrainte c_k où x_i se trouve dans sa portée :
  - Contrainte avec **toutes** les variables **instanciées** : évaluer la contrainte c_k
  - Contrainte avec des variables **non-instanciées** : algorithmes de **prospection**

    ➥ Réduction des domaines des variables **futures**

Architecture d'un solveur de PPC

4. Validation

1. **Succès :**
   - **Empiler** les choix faits par l'hypothèse
   - Retourner à l'étape (1.) s'il reste des **variables à instancier**
   - Enregistrer la **solution** dans **le cas contraire**
   - **Retour en arrière** pour **continuer** l'exploration de l'**arbre**
2. **Échec :**
   - Choisir une **nouvelle valeur** s'il reste des valeurs **non explorées** pour la **variable de l'hypothèse**
   - **Restaurer le contexte** en dépilant le dernier choix fait par l'hypothèse
   - Rechercher une **nouvelle valeur** pour l'**hypothèse précédente**

Prochaine séance

Encore plus sur la résolution :

1. Algorithmes complets de recherche de solutions
2. Techniques d'exploitation de contraintes
3. Techniques rétrospectives et prospectives

Programmation Par Contraintes

Aide à la Décision et Intelligence Artificielle

Licence 3

Dr. **Abdelkader OUALI**
abdelkader.ouali@unicaen.fr

Département Informatique
UFR des sciences
Université de Caen Normandie

2021

Sommaire

- Recherche systématique d'une solution :
  - Générer et Tester **GT**
  - BackTrack (Simple Retour Arrière)
- Techniques prospectives :
  - Forward checking **FC**
  - Looking ahead **LA**

Générer et Tester (**GT**)

Algorithme **complet naïf** de recherche de solutions

Initialement, **aucune instanciation n'est marquée**

1. On génère une **instanciation complète non marquée**

   ➥ $\mathcal{I} = \{(x_1, v_1),\ (x_2, v_2),\ \ldots,\ (x_n, v_n)\}, \quad v_n \in D(x_n)$

2. **Marquer** l'instanciation $\mathcal{I}$
3. **Tester** si les **contraintes** sont **satisfaites** par $\mathcal{I}$
4. Si oui, $\mathcal{I}$ est une **solution**
5. Sinon, **recommencer** depuis 1

Algorithme **GT** (1/2)

Une **fonction récursive booléenne** **GT** pour un CSP $P = (X, D, C)$ dont on doit tester la consistance :

1. P passage par **adresse**
2. $\mathcal{I}$ passage par **valeur**
3. La fonction **GT** est appelée initialement avec $\mathcal{I}$ **vide**, aucune variable n'est instanciée
4. **GT** retourne **vrai** ssi P est **consistant**, sinon elle retourne **faux**

Algorithme **GT** (2/2)

Algorithme 1 : Pseudo-code de **GT**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction GT`

Output : Booléen

Algorithme **GT** (2/2)

Algorithme 2 : Pseudo-code de **GT**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction GT`

Output : Booléen

Fonction `GT`$(\mathcal{I})$
  si $|$`Variables`$(\mathcal{I})|=|X|$ **alors**
    si `Évaluer`$(\mathcal{I})$ **alors**
      retourner Vrai
    sinon
      retourner Faux
  sinon
    Choisir une variable $x_i \in X$ qui **n'est pas encore instanciée**
    pour $v_i \in D(x_i)$ **faire**
      si `GT` $(\mathcal{I} \cup (x_i, v_i))$ **alors**
        retourner Vrai
    retourner Faux

Algorithme **GT** (2/2)

Algorithme 3 : Pseudo-code de **GT**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction GT`

Output : Booléen

Fonction `GT`$(\mathcal{I})$
- **si** $|$`Variables`$(\mathcal{I})|=|X|$ **alors**
  - **si** `Évaluer`$(\mathcal{I})$ **alors**
    - **retourner** Vrai
  - **sinon**
    - **retourner** Faux
- **sinon**
  - Choisir une variable $x_i \in X$ qui **n'est pas encore instanciée**
  - **pour** $v_i \in D(x_i)$ **faire**
    - **si** `GT` $(\mathcal{I} \cup (x_i, v_i))$ **alors**
      - **retourner** Vrai
  - **retourner** Faux

Fonction `Évaluer`$(\mathcal{I})$
- **pour** $c \in C$ **faire**
  - **si** $\neg$ c.`Satisfait`$(\mathcal{I})$ **alors**
    - **retourner** Faux
- **retourner** Vrai

Générer et Tester

inconvénient majeur :

- Parcours **exhaustif** de toutes les instanciations possibles
  ➥ Borne supérieure : $|\max_{1 \leq i \leq n} D(x_i)|^n$ possibilités

1. Si **inexistence** de solutions

2. **Unique solution** consistant en **la toute dernière** instanciation

BackTrack

- **Retour en arrière** est appliqué quand la valeur choisie pour la variable en cours d'instanciation **n'est pas validée** avec l'instanciation partielle sur l'**ensemble de contraintes possibles**
  - Continuer sur une valeur dans le domaine
  - Si toutes les valeurs sont explorées :
    - Retourner **échec** s'il s'agit de la **1er variable** (racine)
    - Retourner à la **variable précédente** et continuer l'exploration
- Si **toutes** les contraintes à vérifier sont **satisfaites** :
  - Retourner **succès** s'il s'agit de la **toute dernière variable**
  - Passer à la **variable suivante** et continuer l'exploration

Algorithme de **BackTrack**

Algorithme 4 : Pseudo-code de **BackTrack**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe BacktrackSolver`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction BT`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction BT (classe LinkedList dans le TP)`

Output : Solution

Fonction `BT`$(\mathcal{I}, \mathcal{V})$
- `// condition d'arrêt de la récurrsivité`
- **si** $\mathcal{V} = \emptyset$ **alors**
  - **retourner** $\mathcal{I}$
- `// choisir une variable non encore instanciée`
- $x_i \leftarrow$ `Retirer`$(\mathcal{V})$
- `// choisir une valeur dans le domaine de` x_i
- **pour** $v_i \in D(x_i)$ **faire**
  - $\mathcal{N} \leftarrow \mathcal{I} \cup (x_i, v_i)$
  - **si** `IsConsistent`$(\mathcal{N})$ **alors**
    - $\mathcal{R} \leftarrow$ `BT`$(\mathcal{N}, \mathcal{V})$
    - **si** $\mathcal{R} \neq Nul$ **alors**
      - **retourner** $\mathcal{R}$
- `Mettre`$(\mathcal{V}, x_i)$
- **retourner** Nul

Fonction `IsConsistent`$(\mathcal{N})$
- **pour** $c \in C$ **faire**
  - **si** `Portée`$(c) \subseteq$ `Variables`$(\mathcal{N})$ **alors**
    - **si** $\neg$ c.`Satisfait`$(\mathcal{N})$ **alors**
      - **retourner** Faux
- **retourner** Vrai

BackTrack

TP :
À partir d'une instanciation vide, appeler la fonction BT() dans la méthode solve() de la classe BacktrackSolver.

Avantage :

- Espace de recherche est **réduit** uniquement sur des instanciations

  partielles satisfaisant les **contraintes** avec les variables déjà instanciées

Inconvénient :

- **Détection des conflits** reste **tardive**

  ➥ On peut faire mieux avec des **techniques prospectives**

Forward Checking **FC**

- **Supprimer** des domaines des variables **non encore instanciées** les **valeurs** qui ne sont **pas compatibles** avec la **valeur choisie** (anticiper)

- *ED* : un argument de plus pour suivre l'**évolution des domaines**

  ➥ passage par **valeur**

Algorithme de **FC**

Algorithme 5 : Pseudo-code de **FC**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre d'une classe`
- Une instantiation partielle $\mathcal{I}$ `// arugment de la fonction FC`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction BT (classe LinkedList dans le TP)`
- Suivre l'évolution des domaines *ED* `// arugment de la fonction FC`

Output : Booléen

Algorithme de **FC**

Algorithme 6 : Pseudo-code de **FC**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre d'une classe`
- Une instantiation partielle $\mathcal{I}$ `// arugment de la fonction FC`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction BT (classe LinkedList dans le TP)`
- Suivre l'évolution des domaines ED `// arugment de la fonction FC`

Output : Booléen

Fonction `FC`$(\mathcal{I}, \mathcal{V}, ED)$

```
// condition d'arrêt de la récurrsivité
si V = ∅ alors
    retourner I
sinon
    // choisir une variable non encore instanciée
    x_i ← Retirer(V)
    pour v_i ∈ ED(x_i) satisfaisant les contraintes unaire sur x_i faire
        V' ← V
        D' = ED ; D'(x_i) = {v_i} ; domaineVide=Faux

        tant que V' ≠ ∅ ∧ ¬domaineVide faire
            // Considérer une variable x_j de l'ensemble V'
            x_j ← Retirer(V')
            D(x_j) = {v_j ∈ ED(x_j) | {(x_i, v_i), (x_j, v_j)} satisfait les contraintes}
            si D'(x_j) = ∅ alors domaineVide=Vrai

        R ← FC(I ∪ {(x_i, v_i)}, V', D')
        si ¬domaineVide ∧ R ≠ Nul alors
            retourner R

    Mettre(V, x_i)
    retourner Nul
```

Forward Checking

Inconvénient :

- Ne supprime **que les valeurs incompatibles** avec la valeur choisie pour la variable **en cours d'instanciation**

- Il peut y avoir des x_i et x_j **non encore instanciées** où $ED(x_i)$ peut avoir des valeurs n'ayant plus de **support** dans $ED(x_j)$

Look Ahead

- Filtrage avec un algorithme de consistance d'arc
- Filtrage durant la recherche récursive d'une solution
  - Avant le début effectif de la recherche (prétraitement)
  - Après chaque instanciation

Prochaine séance :

1. Consistance de nœud
2. Arc consistance
3. Algorithmes d'arc consistance
4. Heuristiques

Programmation Par Contraintes

Aide à la Décision et Intelligence Artificielle

Licence 3

Dr. **Abdelkader OUALI**
abdelkader.ouali@unicaen.fr

Département Informatique
UFR des sciences
Université de Caen Normandie

2020

Sommaire

- Algorithmes de cohérence locale :
  - Cohérence de nœud
  - Cohérence d'arc (AC1 et AC3)
- Heuristiques :
  - Choix de variables
  - Choix de valeurs

Cohérence locale

- **Incomplets en général** mais de **complexité polynomiale**
- **Cohérence de nœud :** cohérence sur des contraintes unaires
- **Cohérence d'arc :** cohérence sur des contraintes unaires et binaires
- **Cohérence de chemin :** cohérence sur des contraintes unaires et binaires

Cohérence de nœud **NC**

Cohérence de nœud (Node Consistency)

- Un CSP $P = (X, D, C)$ est cohérent de noeud si pour toute variable $x_i \in X$, et pour toute valeur $v_i \in D(x_i)$:

  ➠ l'**instanciation partielle** $\{(x_i, v_i)\}$ satisfait toutes les contraintes **unaires** dans C portant sur x_i
- Aucun domaine n'est vide

Principe

- **Supprimer** de $D(x_i)$ toute valeur v_i telle que l'instanciation partielle (x_i, v_i) **viole** les contraintes **unaires portant exclusivement** sur x_i

Algorithme de **NC**

Algorithme 1 : Pseudo-code de **NC**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines ED

Output :

- Les domaines dans ED vérifient la cohérence de noeud
- La fonction retourne Vrai si aucun domaine n'est vide, sinon elle retourne Faux

Fonction `enforceNodeConsistency(`ED`)`

```
pour toute variable x ∈ X faire
    pour toute valeur v ∈ ED(x) faire
        pour toute contrainte unaire c ∈ C faire
            si ¬ c.Satisfait((x, v)) alors
                ED(x) ← ED(x) \ {v}        // Supprimer v du domaine de x
pour toute variable x ∈ X faire
    si ED(x) = ∅ alors retourner Faux
retourner Vrai
```

Support d'une valeur

- Soit une contrainte binaire c portant sur $x_i, x_j \in X$
- La valeur (x_j, v_j) est **support** de la valeur (x_i, v_i) pour la contrainte c si et seulement si (v_i, v_j) **satisfait** c
- Une valeur est **viable** si et seulement si elle possède **au moins un support pour chacune** des contraintes portant sur elle

Support d'une valeur

- Soit une contrainte binaire c portant sur $x_i, x_j \in X$
- La valeur (x_j, v_j) est **support** de la valeur (x_i, v_i) pour la contrainte c si et seulement si (v_i, v_j) **satisfait** c
- Une valeur est **viable** si et seulement si elle possède **au moins un support pour chacune** des contraintes portant sur elle

Par exemple :

- $X : \{x_1, x_2, x_3\}, D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3\}$
- $C = \{x1 < x2 \ et \ x2 = x3\}$

- $(x_2, 2)$ et $(x_2, 3)$ sont **support** de $(x_1, 1)$ pour $x_1 < x_2$ donc $(x_1, 1)$ est **viable**
- $(x_2, 3)$ est **support** de $(x_1, 2)$ pour $x1 < x2$ donc $(x_1, 2)$ est **viable**
- il n'y a pas de support de $(x_1, 3)$ pour $x1 < x2$ donc $(x_1, 3)$ est **non viable**

Arc Cohérence **AC**

Arc cohérence (Arc Consistency)

- Un CSP $P = (X, D, C)$ est arc cohérent si :
  - Il vérifie la cohérence de nœud
  - Pour tout couple de variables (x_i, x_j), et pour toute valeur $v_i \in D(x_i)$, l'instanciation $\{(x_i, v_i)\}$ est viable pour x_j sur toutes les contraintes binaires de C portant exclusivement sur x_i et x_j

Principe

- Rendre le CSP cohérent de nœud
- **Supprimer** de $D(x_i)$ toute valeur v_i **non viable** sur des contraintes binaires portant sur la variable x_i et une variable x_j

Algorithmes de **NC**

- **Revise**
  - Réduit la taille des domaines
  - Supprime les valeurs non viables
- **AC1** :
  - Applique autant que possible **Revise** à tous les arcs sur lesquels il y a une contrainte
  - Si aucun domaine n'a été modifié, la procédure prend fin
  - Réapplique **Revise** à tous les arcs, même aux arcs non modifiés par la passe précédente
- **AC3 :**
  - Ne réapplique **Revise** qu'aux arcs dont le domaine de la variable extrémité **a été modifié**
    ➥ (x_k, x_i) tel que $D(x_i)$ modifié

Algorithme de **Revise**

Algorithme 2 : Pseudo-code de **Revise**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour la 1er variable x_i
- Un argument pour la 2ème variable x_j
- Un argument pour suivre l'évolution des domaines ED

Output :

- Les domaines dans ED vérifient l'arc-cohérence
- La fonction retourne *Vrai* si au moins un domaine des variables est réduit, sinon elle retourne *Faux*

Fonction `REVISE`(x_i, x_j, ED)

```
Del ← Faux
pour toute v_i ∈ ED(x_i) faire
    viable ← Faux
    pour toute v_j ∈ ED(x_j) faire
        toutSatisfait ← Vrai
        pour toute contrainte binaire c ∈ C portant sur x_i et x_j faire
            U ← {(x_i, v_i), (x_j, v_j)}
            si ¬ c.Satisfait(N) alors
                toutSatisfait ← Faux
                break
        75 si toutSatisfait alors
            viable ← Vrai
            break
    si ¬ viable alors
        ED(x_i) ← ED(x_i)\{v_i}          // Supprimer v_i du domaine de x_i
        Del ← Vrai
retourner Del
```

Algorithme de **AC1**

Algorithme 3 : Pseudo-code de **AC1**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines *ED*

Output :

- Les domaines *ED* sont arc-cohérent si aucun domaine n'est vide
- La fonction retourne *Faux* si au moins un domaine est vide, sinon elle retourne *Vrai*

Fonction `AC1(`*ED*`)`
  si ¬`NC(`*ED*`)` **alors**
    retourner Faux

  faire
    change ← *Faux*
    pour tout *couple* (x_i, x_j) *dans* X **faire**
      si `REVISE(`x_i, x_j, *ED*`)` **alors**
        change ← *Vrai*
  tant que *change* = *Vrai*

  pour *toute variable* $x \in X$ **faire**
    si $ED(x) = \emptyset$ **alors retourner** Faux
  retourner Vrai

Algorithme de **MAC** solver

Algorithme 4 : Pseudo-code de **MAC**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe BacktrackSolver`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction MAC`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction MAC (classe LinkedList dans le TP)`
- Suivre l'évolution des domaines ED `// arugment de la fonction MAC`

Output : Solution

Fonction `MAC`$(\mathcal{I}, \mathcal{V}, ED)$

```
// conditions d'arrêt de la récurrsivité
si V = ∅ alors
    retourner I
sinon
    // Réduction des domaines des variables par l'arc-cohérence
    si ¬ AC1(X, ED, C) alors
        retourner Nul
    // choisir une variable non encore instanciée
    x_i ← Retirer(V)
    // choisir une valeur dans le domaine de x_i
    pour v_i ∈ ED(x_i) faire
        N ← I ∪ (x_i, v_i)
        si IsConsistent(N) alors
            R ← MAC(N, V, ED)
            si R ≠ Nul alors
                retourner R
    Mettre(V, x_i)
    retourner Nul
```

Algorithme de **AC3** (n'est pas forcément requis pour le TP)

Algorithme 5 : Pseudo-code de **AC3**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines ED

Output :

- Les domaines ED sont arc-cohérent si aucun domaine n'est vide
- La fonction retourne *Faux* si au moins un domaine est vide, sinon elle retourne *Vrai*

Fonction `AC3(`ED`)`

si $\neg$`NC(`ED`)` **alors**
  retourner Faux
$Q \leftarrow \{(x_i, x_j) : (i \neq j) \wedge (\text{ il existe une contrainte entre } x_i \text{ et } x_j)\}$
tant que $Q \neq \emptyset$ **faire**
  $Q \leftarrow Q \setminus \{(x_i, x_j)\}$ `// Prendre une paire de variables` (x_i, x_j) `de` Q
  si `REVISE(`x_i, x_j, ED`)` **alors**
    `// il y a eu réduction du domaine de` x_i
    $Q \leftarrow Q \cup \{(x_k, x_i) : \text{il existe une contrainte entre } x_k \text{ et } x_i \text{ et } x_k \neq x_i \text{ et } x_k \neq x_j\}$
pour *toute variable* $x \in X$ **faire**
  si $ED(x) = \emptyset$ **alors retourner** Faux
retourner Vrai

Heuristiques

Heuristique

Définition

Méthode permettant en pratique d'obtenir plus rapidement une solution, très souvent non optimale.

- Choix de la **prochaine variable**
- Choix de la **prochaine valeur**

Choix de variable

Information sur la variable :

- Taille de son domaine
- Le nombre de contraintes qui l'utilise
- Première/Dernière
- Celle avec le domaine le plus petit/grand
- Celle dans le plus petit/grand nombre de contraintes
- |domaine| / |contraintes|

Choix de valeur

Information sur la valeur :

- Première/Dernière
- Au milieu
- Alternance début/fin

Choix de valeur

Pourquoi c'est important ?

- Ne change pas la taille de l'arbre de recherche
- Pas d'effet sur l'énumération des solutions
- Mais si on cherche une solution, impact sur le temps nécessaire pour l'obtenir

Hasard

Choix aléatoire :

- Souvent le hasard fait bien les choses
- Peut avoir des gains spectaculaires
- Choisir au hasard l'ordre des variables
- Choisir au hasard en cas d'ex aequos pour les heuristiques
- Choisir une heuristique au hasard

Fin