

Rappel du cours précédent

```
listsum([],X) :- {X=0}.
listsum([H|R],X) :- {X = H + XR}, listsum(R,XR).

listsum([2, Y], 5)           | ⊤
{5 = 2 + XR}, listsum([Y], XR) | ⊤
listsum([Y], XR)             | XR = 3
{XR = Y + XR'}, listsum([], XR') | XR = 3
listsum([], XR')             | Y = 3 - XR' ∧ XR = 3
{XR' = 0}                    | Y = 3 - XR' ∧ XR = 3
                             | Y = 3 ∧ XR = 3 ∧ XR' = 0
```

La réponse est la projection de la contrainte résolue aux variables de la requête: $Y = 3$.

Contraintes sur un domaine fini

- ▶ On a pour toute variable un *choix fini* de ses valeurs.
- ▶ Une classe importante de domaine de contraintes.
- ▶ Utilisée pour modéliser des problèmes avec des choix.
- ▶ Ordonnancement, Emploi du temps, routage, etc.
- ▶ Beaucoup d'applications industrielles.

Plan de ce cours

- Contraintes sur un domaine fini.
- ▶ Solutionneur "génère et teste"
 - ▶ Solutionneur par retour en arrière
 - ▶ Consistance d'arc et de noeuds
 - ▶ Heuristiques
 - ▶ Consistance de bornes
 - ▶ Consistance généralisée

Dans ce cours on parle seulement du *solutionneur*.

Problème de satisfaction de contraintes

- ▶ Une contrainte C sur des variables x_1, \dots, x_n
- ▶ Un domaine $D(x_i)$ pour chaque variable
- ▶ Une contrainte C est implicitement donnée par
$$C \wedge x_1 \in D(x_1) \wedge \dots \wedge x_n \in D(x_n)$$
- ▶ Contrainte *binaires* : Les contraintes primitives contiennent au plus deux variables. Donne lieu à un *graphe de contraintes*.
- ▶ On écrit « $D(x) = \{c_1, \dots, c_n\}$ » au lieu de « $x \in \{c_1, \dots, c_n\}$ ».

Exemple: Colorer une carte

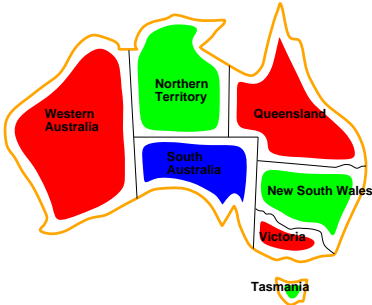
Trois couleurs, des regions adjacentes doivent avoir de couleurs différentes.



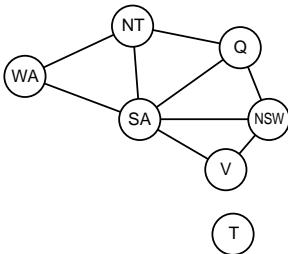
$$WA \neq NT \wedge WA \neq SA \wedge NT \neq SA \wedge NT \neq Q \wedge SA \neq Q \wedge SA \neq V \wedge Q \neq NSW \wedge NSW \neq V$$
$$D(WA) = D(NT) = D(SA) = D(Q) = D(V) = D(NSW) = D(T) = \{rouge, jaune, bleu\}$$



Exemple: Colorer une carte



Graphe de contraintes



Exemple: Les 4 reines

- Placer 4 reines sur un échiquier de taille 4x4 de sorte qu'aucune reine est en prise
- Quatre variables Q_1, Q_2, Q_3, Q_4 qui représentent la ligne de la reine dans chaque colonne. Domaine de chaque variable: $\{1, 2, 3, 4\}$
- Les contraintes: $Q_1 \neq Q_2 \wedge Q_1 \neq Q_3 \wedge Q_1 \neq Q_4 \wedge Q_2 \neq Q_3 \wedge Q_2 \neq Q_4 \wedge Q_3 \neq Q_4$
- $Q_1 \neq Q_2 + 1 \wedge Q_1 \neq Q_3 + 2 \wedge Q_1 \neq Q_4 + 3 \wedge Q_2 \neq Q_3 + 1 \wedge Q_2 \neq Q_4 + 2 \wedge Q_3 \neq Q_4 + 1$
- $Q_1 \neq Q_2 - 1 \wedge Q_1 \neq Q_3 - 2 \wedge Q_1 \neq Q_4 - 3 \wedge Q_2 \neq Q_3 - 1 \wedge Q_2 \neq Q_4 - 2 \wedge Q_3 \neq Q_4 - 1$



Exemple: sac du contrebandier

Solutionneur génère et teste

- ▶ Contrebandier avec un sac de capacité 9.
- ▶ Il doit choisir des objets pour faire un profit d'au moins 30

objet	profit	poids
whisky	15	4
parfum	10	3
cigarettes	7	2

$4W + 3P + 2C \leq 9 \wedge 15W + 10P + 7C \geq 30$

- ▶ Domaines des variables ?

- ▶ Le plus simple est d'énumérer les affectations possibles.
- ▶ Le solutionneur **génère et teste** :
 - ▶ Énumère une par une les valeurs des variables une par une
 - ▶ Quand *chaque* variable a une valeur, on teste si la contrainte est satisfaite ou pas.
- ▶ Très inefficace !
- ▶ On peut améliorer cette technique en testant à chaque fois, si l'affectation partielle entraîne déjà la non-satisfaisabilité : voir le transparent suivant.

Solutionneur simple par retour en arrière

Solutionneur par retour en arrière

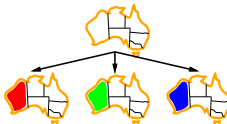
- ▶ Le solutionneur simple par retour en arrière:
 - ▶ énumère une par une les valeurs des variables une par une
 - ▶ vérifie qu'aucune contrainte simple est fausse à chaque étape
 - ▶ On peut facilement tester la satisfaisabilité d'une contrainte simple sans variables.
 - ▶ *partsat*(*C*) retourne faux, si *C* n'est pas satisfaisable à cause d'une contrainte simple sans variables qui n'est pas satisfaisable. Sinon *partsat*(*C*) retourne vrai.

- partsat*(*C*) = vraie ssi toute contrainte simple et close de *C* est vraie
- backsolve*(*C*, *D*)
- ▶ Si *variables*(*C*) est vide, alors retourne *partsat*(*C*)
 - ▶ Choisir *x* dans *variables*(*C*)
 - ▶ Pour chaque valeur *d* dans *D*(*x*)
 - ▶ Soit *C*₁ la contrainte *C* où *x* est remplacé par *d*
 - ▶ Si *partsat*(*C*₁) alors *backsolve*(*C*₁, *D*) retourne vraie
 - ▶ Retourne faux

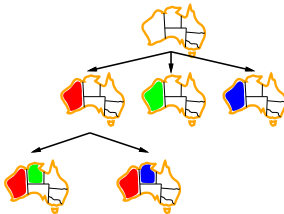
Exemple retour en arrière



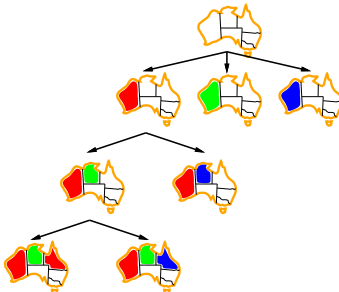
Exemple retour en arrière



Exemple retour en arrière



Exemple retour en arrière



Complexité des contraintes sur un domaine fini

- ▶ Satisfaisabilité de contraintes sur un domaine fini est NP-complet (même si on se restreint à des domaines de cardinalité 2).
- ▶ C'est-à-dire :
 - ▶ On ne connaît que des algorithmes avec complexité exponentielle au pire des cas.
 - ▶ Il est fort probable qu'il n'y a pas d'algorithme avec une meilleure complexité.
- ▶ Voir un cours de *calculabilité et complexité*.



Consistance de noeud et d'arc

- ▶ Idée: Trouver un CSP équivalent au CSP d'origine qui a des domaines de variables plus petits.
- ▶ C'est le principe de la *simplification de contraintes*, appliqué aux domaines des variables.
- ▶ On considère les contraintes simples une par une.
- ▶ Consistance de noeud : ($variables(c) = \{x\}$): enlever chaque valeur du domaine de x qui rend c insatisfaisable.
- ▶ Consistance d'arc : ($variables(c) = \{x, y\}$): enlever chaque valeur de $D(x)$ pour laquelle il n'y a pas de valeur dans $D(y)$ qui satisfait c et vice-versa.



Consistance de noeud

- ▶ Une contrainte simple c est noeud-consistante avec domaine D , si
 - ▶ soit $|\text{variables}(c)| \neq 1$;
 - ▶ soit $\text{variables}(c) = \{x\}$, et pour chaque d dans $D(x)$, $\{x \leftarrow d\}$ est une solution de c .
- ▶ Un CSP est noeud-consistant, si chaque contrainte simple est noeud-consistante.



Comment obtenir un CSP noeud-consistant ?

$$\text{noeudcons}(C, D)$$

- Pour chaque contrainte simple c dans C
 - $D := \text{noeudconssimple}(c, D)$
- retourne D

$$\text{noeudconssimple}(c, D)$$

- ▶ Si $|variables(c)| = 1$ alors
 - ▶ Soit $\{x\} = variables(c)$
 $D(x) := \{d \in D(x) \mid \{x \leftarrow d\} \text{ est une solution de } c\}$
- ▶ retourne D



Arc consistance

Comment obtenir un CSP arc-consistant ?

- ▶ Une contrainte simple est arc-consistante avec domaine D , si
 - ▶ soit $|variables(c)| \neq 2$;
 - ▶ soit $variables(c) = \{x, y\}$ et
 - ▶ pour chaque d dans $D(x)$, il y a $e \in D(y)$ tel que $\{x \leftarrow d, y \leftarrow e\}$ est une solution de c
 - ▶ et analogue pour y .
- ▶ Un CSP est arc-consistant, si chaque contrainte simple est arc-consistante.

- $arconssimple(c, D)$
- ▶ si $|variables(c)| = 2$ alors
 - ▶ $D(x) := \{d \in D(x) | \exists e \in D(y) \text{ t.q. } \{x \leftarrow d, y \leftarrow e\} \text{ est une solution de } c\}$
 - ▶ $D(y) := \{e \in D(y) | \exists d \in D(x) \text{ t.q. } \{x \leftarrow d, y \leftarrow e\} \text{ est une solution de } c\}$
 - ▶ retourne D

Enlève des valeurs non arc-consistantes avec c

Comment obtenir un CSP arc-consistant ?

Arc-consistance : calcul d'un point fixe

- $arcons(C, D)$
- ▶ Répète
 - ▶ $W := D$
 - ▶ Pour chaque contrainte simple c de C
 - ▶ $D := arconssimple(c, D)$
 - ▶ jusqu'à $W = D$
 - ▶ retourne D

- ▶ Remarquer la boucle : le traitement d'une contrainte primitive peut déclencher qu'une autre contrainte primitive est à traiter de nouveau.
- ▶ Exemple:

$$X_1 < X_2 \wedge X_2 < X_3 \wedge X_3 < X_4$$
$$X_1, X_2, X_3, X_4 \in \{0, 1, 2, 3, 4\}$$

(sera fait au tableau)

Utiliser noeud et arc consistance

- ▶ On peut définir des solveurs.
- ▶ Deux types de domaines importants :
 - ▶ domaine *faux* : une variable a un domaine vide
 - ▶ domaine *simple* : toutes les variables ont un domaine singleton (de taille un)
- ▶ Étendre satisfaisabilité sur des CSPs avec des domaines simples.

Solveur noeud et arc consistance

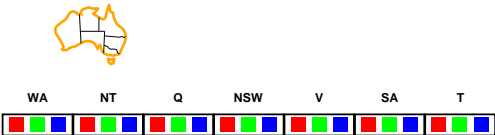
- Solveur **incomplet**
- ▶ $D := \text{noeudcons}(C, D)$
 - ▶ $D := \text{arccons}(C, D)$
 - ▶ Si D est un domaine faux, alors retourne *faux*
 - ▶ Si D est un domaine simple, alors retourne *satisfaisable*(C, D)
 - ▶ sinon retourne *inconnu*

Comment définir un solveur **complet** en utilisant arc et noeud consistance ?

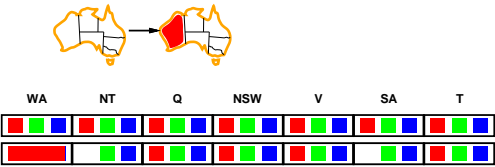
Retour en arrière avec consistance

- ▶ Combiner le solveur par retour en arrière avec consistance.
- ▶ Appliquer noeud (et/ou) arc consistance avant de lancer le solveur par retour en arrière **et** après chaque fois qu'une variable est affectée par le solveur.

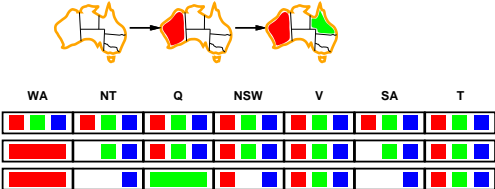
Exemple avec noeud consistance uniquement



Exemple avec noeud consistance

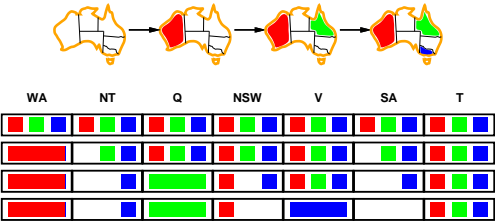


Exemple avec noeud consistance



Ce problème est-t-il arc-consistant ?

Exemple avec noeud consistance



Heuristiques

- On peut utiliser des heuristiques pour choisir la variable à affecter et la valeur.
- statique/dynamique

La variable la plus contrainte

- Choisir la variable avec le plus petit nombre de valeurs légales



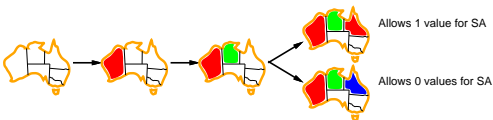
La variable la plus contraignante

- ▶ En cas d'égalité pour la variable la plus contrainte
- ▶ Choisir la variable qui a le plus de contraintes avec les variables qui restent



La valeur la moins contraignante

- Pour une variable, choisir la valeur qui contraint le moins possible les variables qui restent



Consistance pour des contraintes avec plus de 2 variables

- ▶ Quoi faire avec des contraintes avec plus de 2 variables ?
- ▶ Consistance d'hyper-arc: étendre l'arc consistance à un nombre arbitraire de variables
- ▶ Déterminer la hyper-arc consistance est NP-difficile



Consistance de bornes

- ▶ CSP arithmétique: les contraintes sont sur des entiers
- ▶ intervals: $[l..u]$ représente l'ensemble $\{l, l+1, \dots, u\}$
- ▶ Idée: Utiliser la consistance sur les réels et examiner seulement les bornes (inférieurs et supérieurs) du domaine de chaque variable
- ▶ Définir $\min(D, x)$ comme l'élément minimum dans le domaine de x , pareil $\max(D, x)$



Consistance de bornes

- ▶ Une contrainte simple c est bornes-consistante avec domaine D , si pour chaque variable x dans $variables(c)$
 - ▶ ils existent des réels d_1, \dots, d_k pour les autres variables x_1, \dots, x_k tel que
 - ▶ $\min(D, x_j) \leq d_j \leq \max(D, x_j)$ pour tout j et
 - ▶ $\{x \leftarrow \min(D, x), x_1 \leftarrow d_1, \dots, x_k \leftarrow d_k\}$ est une solution de c
 - ▶ ils existent des réels d'_1, \dots, d'_k pour les autres variables x_1, \dots, x_k tel que
 - ▶ $\min(D, x_j) \leq d'_j \leq \max(D, x_j)$ pour tout j et
 - ▶ $\{x \leftarrow \max(D, x), x_1 \leftarrow d'_1, \dots, x_k \leftarrow d'_k\}$ est une solution de c
- ▶ Un CSP arithmétique est bornes-consistant, si toutes ses contraintes simples le sont



Comment obtenir un CSP bornes-consistant ?

- ▶ Étant donné un domaine D , on doit modifier les bornes, de sorte que le résultat est bornes-consistant
- ▶ Utilisation de règles de propagation
- ▶ Exemple:
 - ▶ $X = Y + Z$ équivalent à $Y = X - Z$ et $Z = X - Y$
 - ▶ Raisonner avec \max et \min
 - ▶ $X \geq \min(D, Y) + \min(D, Z)$, $X \leq \max(D, Y) + \max(D, Z)$
 - ▶ $Y \geq \min(D, X) - \max(D, Z)$, $Y \leq \max(D, X) - \min(D, Z)$
 - ▶ $Z \geq \min(D, X) - \max(D, Y)$, $Z \leq \max(D, X) - \min(D, Y)$
 - ▶ cela donne des **règles de propagation**



Example

- ▶ $X = Y + Z$, $D(X) = [4..8]$, $D(Y) = [0..3]$, $D(Z) = [2..2]$
- ▶ Les règles de propagation donnent:
 - ▶ $(0 + 2 =) 2 \leq X \leq 5 (= 3 + 2)$
 - ▶ $(4 - 2 =) 2 \leq Y \leq 6 (= 8 - 2)$
 - ▶ $(4 - 3 =) 1 \leq Z \leq 8 (= 8 - 0)$
- ▶ Les domaines peuvent être réduits:
 $D(X) = [4..5]$, $D(Y) = [2..3]$, $D(Z) = [2..2]$



D'autres règles de propagation

Inégalités $Y \neq Z$

- ▶ $4W + 3P + 2C \leq 9$
- ▶ $W \leq \frac{9}{4} - \frac{3}{4}min(D, P) - \frac{2}{4}min(D, C)$
- ▶ $P \leq \frac{9}{3} - \frac{4}{3}min(D, W) - \frac{2}{3}min(D, C)$
- ▶ $C \leq \frac{9}{2} - \frac{4}{2}min(D, W) - \frac{3}{2}min(D, P)$
- ▶ Étant donné un domaine initial
 $D(W) = [0..9], D(P) = [0..9], D(C) = [0..9]$ on détermine que
 $W \leq \frac{9}{4}, P \leq \frac{9}{3}, C \leq \frac{9}{2},$
- ▶ nouveau domaine:
 $D(W) = [0..2], D(P) = [0..3], D(C) = [0..4]$

- ▶ Les inégalités donnent des règles de propagation très faibles
- ▶ Seulement si une de deux côtés prend une valeur fixe qui est égale au minimum ou maximum de l'autre il y a propagation
- ▶ $D(Y) = [2..4], D(Z) = [2..3]$ pas de propagation
- ▶ $D(Y) = [2..4], D(Z) = [3..3]$ pas de propagation
- ▶ $D(Y) = [2..4], D(Z) = [2..2]$ propagation
 $D(Y) = [3..4], D(Z) = [2..2]$

Multiplication $X = Y * Z$

Multiplication $X = Y * Z$

- ▶ Si toutes les variables sont positives
 $X \geq min(D, Y) * min(D, Z), X \leq max(D, Y) * max(D, Z)$
etc. pour Y, Z
- ▶ sinon
 $X \geq minimum\{min(D, Y)*min(D, Z), min(D, Y)*max(D, Z), max(D, Y) * min(D, Z), max(D, Y) * max(D, Z)\}$
- ▶ similaire pour borne supérieure pour X en utilisant *maximum*

- ▶ Règles de propagation pour Y et Z ?
- ▶ si $min(D, Z) < 0$ et $max(D, Z) > 0$ il n'y a pas de restriction pour Y
- ▶ On "attend" jusqu'à ce que le domaine de Z devienne non-négatif ou non-positif et ensuite en utilise des règles de la forme
$$Y \geq minimum\{min(D, X)/min(D, Z), min(D, X)/max(D, Z), max(D, X)/min(D, Z), max(D, X)/max(D, Z)\}$$

Attention à la division par 0

Algorithme de bornes consistance

- $\text{bornescons}(C, D)$: Appliquer les règles de propagation pour chaque contrainte simple de C , jusqu'à ce qu'il n'y a plus de changement dans les domaines D .
- On ne réexamine pas une contrainte simple, si les domaines de ses variables n'ont pas changé.



Solutionneur bornes consistance

- ▶ $D := \text{bornescons}(C, D)$
- ▶ Si D est un domaine faux, alors retourne *false*
- ▶ Si D est un domaine simple, alors retourne $\text{satisfaisable}(C, D)$
- ▶ sinon retourne *inconnu*



Solutionneur par retour en arrière avec bornes consistance

- ▶ Appliquer bornes consistance avant de lancer le solveur par retour en arrière **et** à chaque fois qu'une variable est affectée par le solveur retour en arrière.



Exemple retour en arrière avec bornes consistance

- ▶ Problème du sac du contrebandier
- ▶ $4W + 3P + 2C \leq 9 \wedge 15W + 10P + 7C \geq 30$
- ▶ Domaines initiaux : $D(W) = [0..9]$, $D(P) = [0..9]$, $D(C) = [0..9]$
- ▶ Bornes consistance sur la première contrainte donne : $D(W) = [0..2]$, $D(P) = [0..3]$, $D(C) = [0..4]$
- ▶ On essaie $W = 0$. Ça donne : $D(W) = [0..0]$, $D(P) = [1..3]$, $D(C) = [0..3]$
- ▶ On essaie $P = 1$. Ça donne : $D(W) = [0..0]$, $D(P) = [1..1]$, $D(C) = [3..3]$ et on a trouvé une solution.
- ▶ On peut aussi chercher les autres solutions



Consistance généralisée

- ▶ On peut combiner les trois consistances (nœud, arc, bornes) vues jusqu'à présent.
- ▶ Toutes ses méthodes utilisent les contraintes simples une par une.
- ▶ On peut considérer des contraintes simples "complexes" qui sont une conjonction de contraintes simples avec un mécanisme de propagation spécial.
- ▶ Exemple: $alldifferent(\{V_1, \dots, V_n\})$
- ▶ $alldifferent(\{X, Y, Z\})$ signifie $X \neq Y \wedge Y \neq Z \wedge X \neq Z$
- ▶ Arc-consistant avec $D(X) = \{1, 2\}$, $D(Y) = \{1, 2\}$, $D(Z) = \{1, 2\}$
- ▶ Mais il n'y a pas de solution.

Consistance pour alldifferent

- ▶ Soit c de la forme $alldifferent(V)$
- ▶ Tant qu'il existe $v \in V$ avec $D(v) = \{d\}$
 - ▶ $V := V - \{v\}$
 - ▶ Pour chaque $v' \in V$
 - ▶ $D(v') := D(v') - \{d\}$
- ▶ $DV := \bigcup_{v \in V} D(v)$
- ▶ Si $|V| > |DV|$ alors retourne *domaine faux*
- ▶ retourne D

Exemples alldifferent

- ▶ $alldifferent(\{X, Y, Z\})$ avec $D(X) = \{1, 2\}$, $D(Y) = \{1, 2\}$, $D(Z) = \{1, 2\}$
- ▶ Algorithme retourne *faux*
- ▶ $alldifferent(\{X, Y, Z, T\})$ avec $D(X) = \{1, 2\}$, $D(Y) = \{1, 2\}$, $D(Z) = \{1, 2\}$, $D(T) = \{2, 3, 4, 5\}$
- ▶ Algorithme ne detecte pas le problème
- ▶ On peut utiliser des algorithmes plus compliqués pour cela.

Exemple d'utilisation de alldifferent

		9			1	6	2	
5	7			2	8		3	
3			7					4
8	9			7		4		
	6		5		3		9	
		1		9			7	6
6				7				8
	4		1	3			6	5
	2	7	6			9		

Sudoku

- ▶ Le problème du Sudoku consiste à remplir une grille de sorte que chaque ligne, chaque colonne et chaque carré contiennent les chiffres 1 à 9.
- ▶ Pour modéliser ce problème on peut utiliser alldifferent. Comment ?