Chapitre 2
La fouille et le filtrage

Claude-Guy Quimper

Processus de résolution

- Comment résoudre un problème de satisfaction de contraintes?
- C'est ce que nous allons tenter de répondre dans ce chapitre.
- Si notre solveur offre suffisamment de contraintes pour encoder un problème NP-Complet, il faudra s'attendre à ce que la complexité de l'algorithme soit exponentielle.

Énumération des solutions candidates

- Une première approche serait d'utiliser la force brute et d'énumérer toutes les solutions candidates possibles.
- On pourrait ensuite itérer sur chacune des solutions candidates et filtrer celles qui ne satisfont pas toutes les contraintes.

Exemple

$dom(X_1) = \{1, 2, 3\}$

$dom(X_2) = \{1, 2\}$

$X_1 \leq X_2$

$X_1 \geq X_2$

X_1	X_2	$X_1 \leq X_2$	$X_1 \geq X_2$
1	1	✓	✓
1	2	✓	✗
2	1	✗	✓
2	2	✓	✓
3	1	✗	✓
3	2	✗	✓

Exemple

$dom(X_1) = \{1, 2, 3\}$

$dom(X_2) = \{1, 2\}$

$X_1 \leq X_2$

$X_1 \geq X_2$

X_1	X_2	$X_1 \leq X_2$	$X_1 \geq X_2$
1	1	✓	✓
1	2	✓	✗
2	1	✗	✓
2	2	✓	✓
3	1	✗	✓
3	2	✗	✓

Analyse de l'énumération

- Peu importe l'algorithme utilisé pour énumérer et tester les solutions, cet algorithme doit passer au moins une unité de temps sur chaque solution candidate.
- Nous obtenons donc une complexité dans

$$\Omega(\prod_{i=1}^{n} |\mathrm{dom}(X_i)|)$$

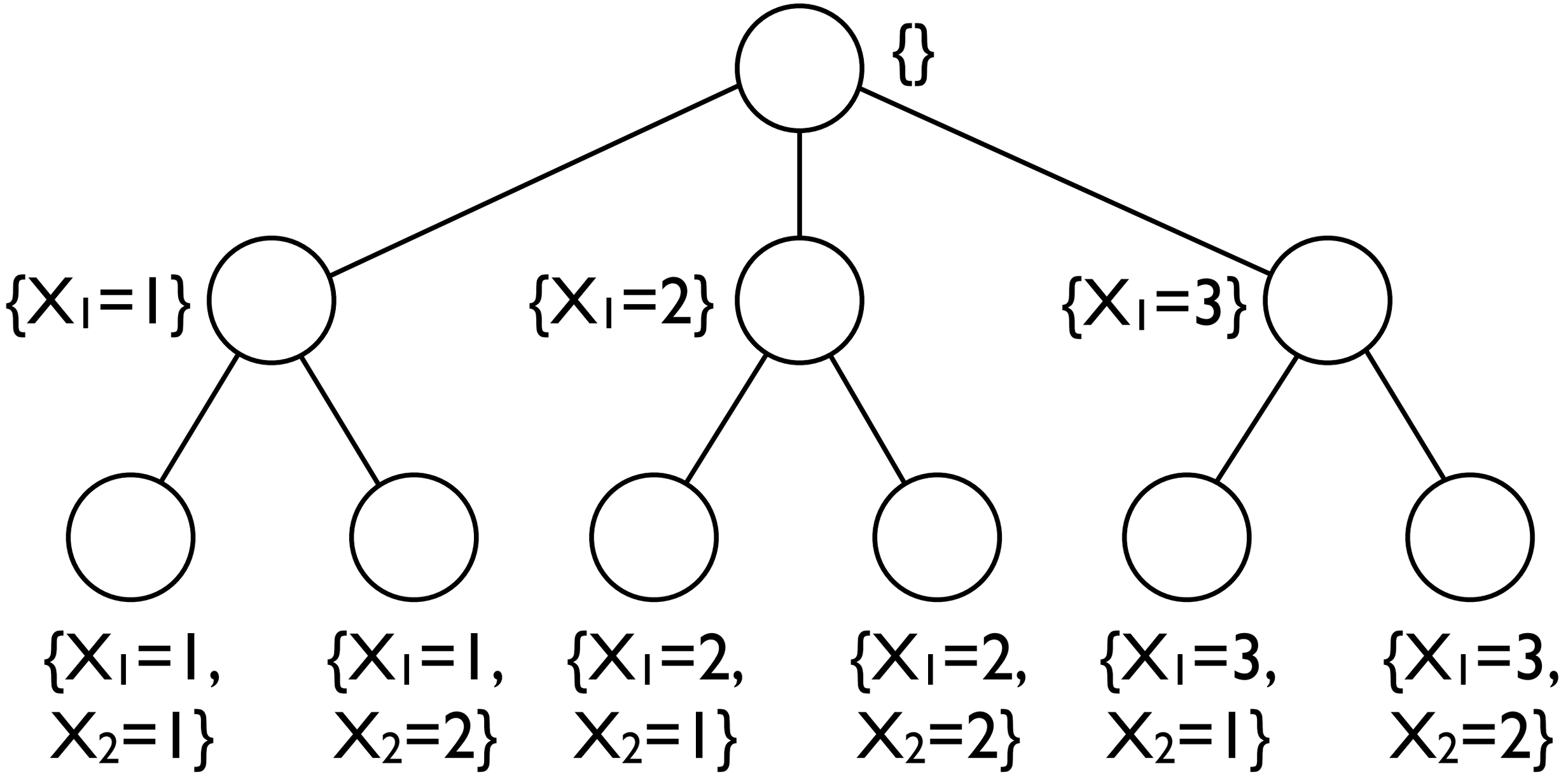
Importance de l'énumération

- Même si cette technique peut sembler naïve, c'est sur l'énumération que sont fondés les algorithmes les plus sophistiqués.
- Nous étudierons donc cette technique avec soins afin de l'améliorer.

Fouille en profondeur

- La fouille en profondeur est une façon efficace d'énumérer toutes les solutions d'un problème.
- Elle consiste à créer un arbre de recherche où chaque noeud représente une solution partielle, c'est-à-dire une solution où seules quelques variables sont assignées à une valeur.
- La racine de l'arbre est une solution partielle vide où aucune variable n'est assignée.
- Chaque noeud hérite de la solution partielle de son parent et y ajoute une affectation de plus.

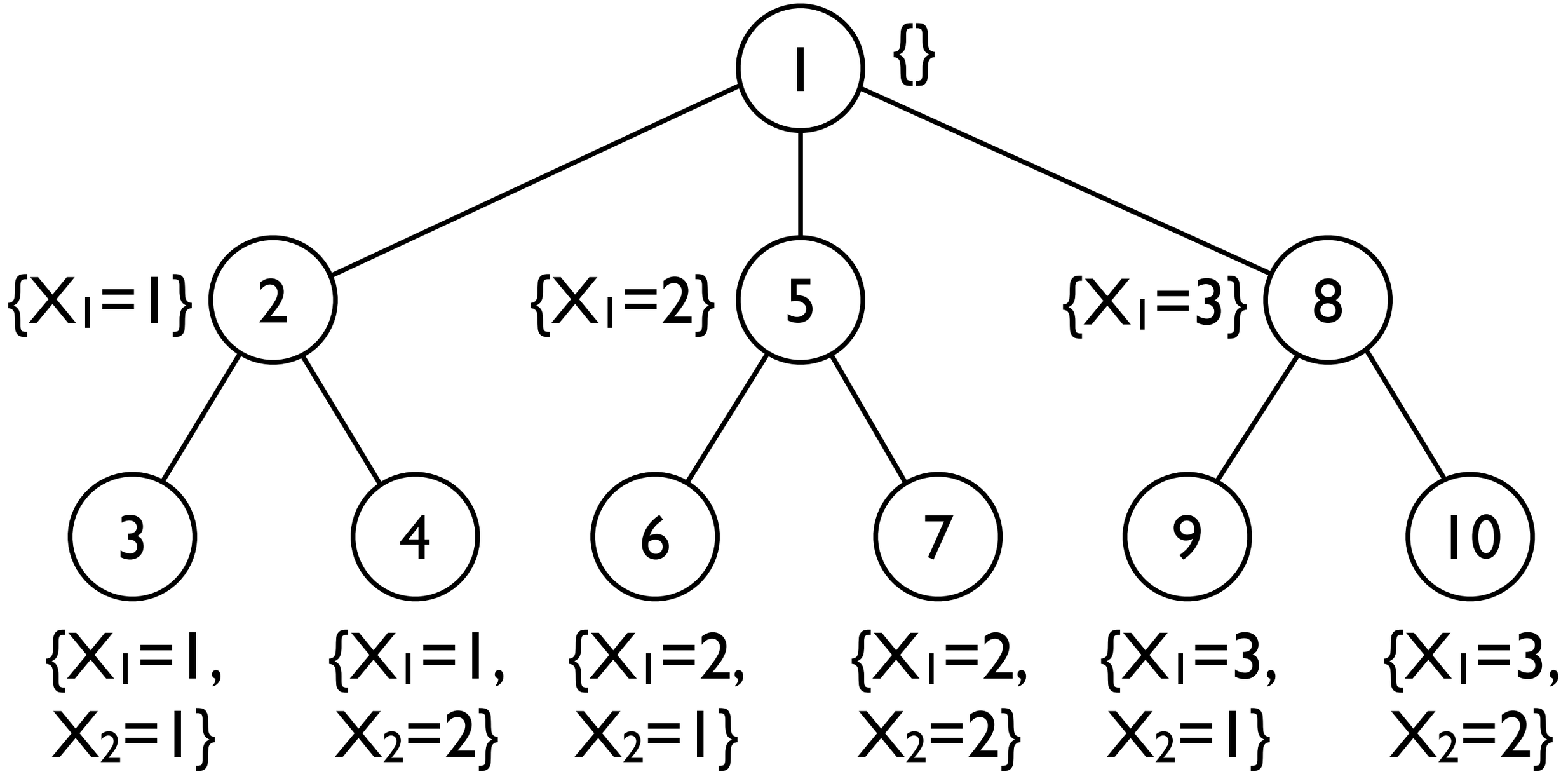
Arbre de recherche



Ordre de visite

- La visite de l'arbre commence à la racine.
- On visite ensuite un enfant du noeud courant qui n'a pas encore été visité en donnant priorité aux noeuds de gauche.
- Si tous les enfants ont été visités, on retourne visiter le parent.

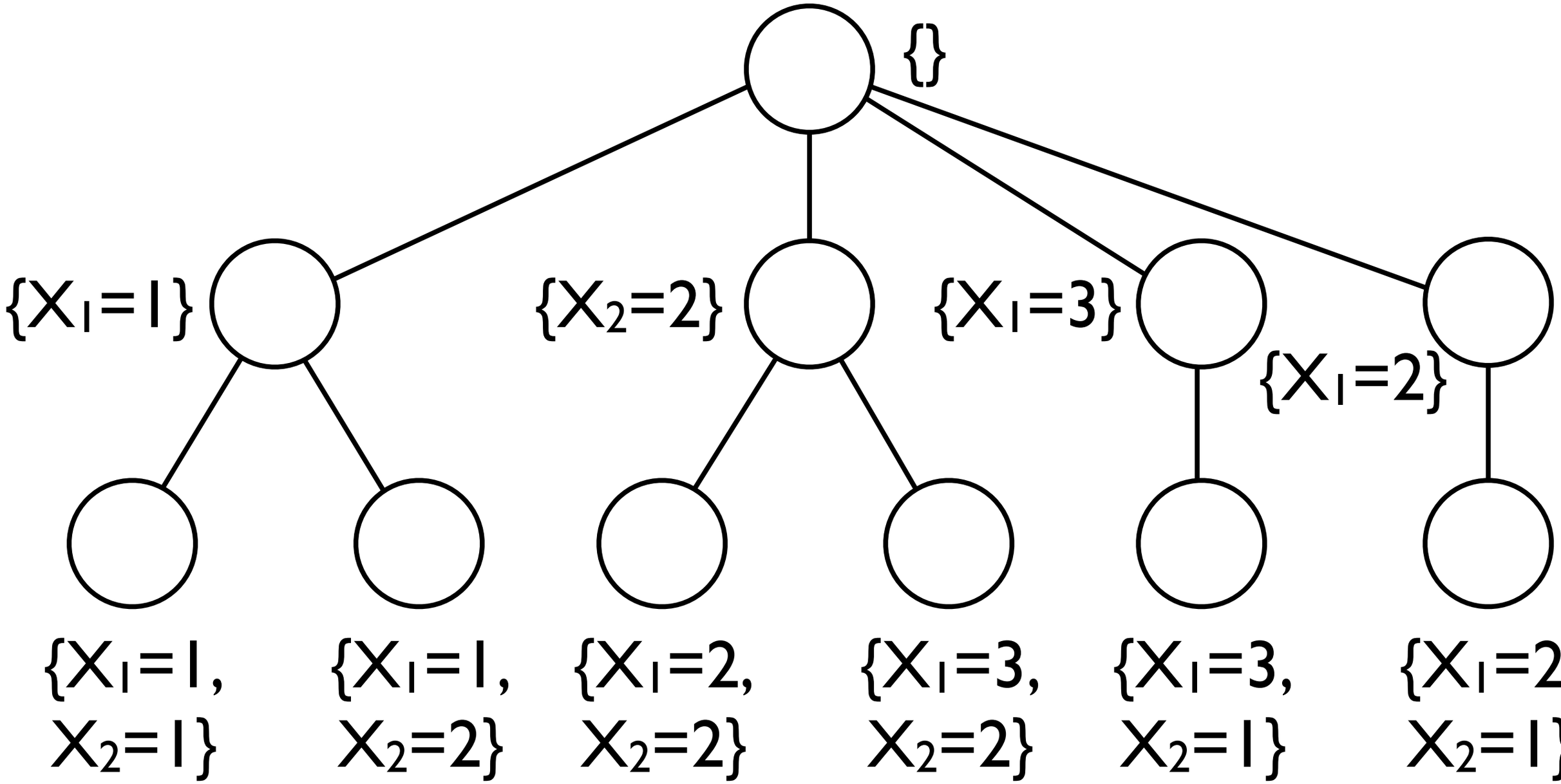
Ordre de visite



Plusieurs arbres de recherche

- Il existe plusieurs arbres de recherche pour un même problème.
- Par exemple, dans l'arbre précédent, nous avons instancié la variable X_1 avant la variable X_2. Nous aurions très bien pu faire l'inverse et instancier X_2 avant X_1.
- Après avoir exploré les solutions avec $X_1 = 1$, nous ne sommes par forcé d'explorer les solutions avec une autre valeur pour X_1. On peut instancier une autre variable que X_1.

Plusieurs arbres de recherche



Validité d'un arbre de recherche

- L'arbre de recherche doit visiter toutes les solutions candidates possibles.
  - Cette condition est nécessaire
- L'arbre de recherche ne doit pas visiter deux fois la même solution candidate ou la même solution partielle.
  - Cette condition n'est pas nécessaire. Cependant, la violer implique une perte en efficacité.

Qu'est-ce qui décide de l'arbre de recherche?

- Ce sont les heuristiques de recherche qui font pousser l'arbre de recherche.
- Une heuristique est une politique de choix qui donne priorité à certaines solutions.
- Le solveur peut alors chercher des solutions dans l'espace de recherche qui lui semble le plus prometteur.
- Dans l'arbre de recherche de la page 13, l'heuristique a « jugé » plus prometteuses les solutions avec $X_1=1$ que les solutions avec $X_2=2$.

Pseudo-code

Algorithme 1 : FouilleEnProfondeurComplète(dom$(X_1), \ldots,$ dom(X_n))

répéter
  $Candidats \leftarrow \{X_i \mid |\text{dom}(X_i)| > 1\}$
  si $Candidats = \emptyset$ **alors**
    // Tous les domaines ne contiennent qu'une seule valeur.
    si *dom*$(X_1), \ldots,$ *dom*(X_n) *satisfait toutes les contraintes* **alors**
      retourner dom$(X_1), \ldots,$ dom(X_n)
    sinon
      retourner $\emptyset$
  Choisir une variable $X_i \in Candidats$
  Choisir une valeur $v \in \text{dom}(X_i)$
  $Solution \leftarrow$ FouilleEnProfondeurComplète(dom$(X_1), \ldots,$ dom$(X_{i-1}), \{v\},$ dom$(X_{i+1}), \ldots,$ dom(X_n))
  si $Solution = \emptyset$ **alors**
    dom$(X_i) \leftarrow$ dom$(X_i) \setminus \{v\}$
jusqu'à $Solution \neq \emptyset$
retourner $Solution$

Comment améliorer la recherche?

- En faisant de bons choix
  - Nous verrons des heuristiques de choix de variable et de choix de valeur qui permettront d'arriver plus tôt à une solution. Pour le moment, contentons-nous de dire que nous faisons les meilleurs choix possibles, mais que ces choix peuvent s'avérer vains.
- En détectant plus tôt dans le processus les branches qui ne mènent à aucune solution.

Première amélioration

- Comme première amélioration, on peut tester une contrainte dès que les variables de sa portée sont instanciées à des valeurs.
- Si une contrainte n'est pas satisfaite dans une solution partielle, elle ne le sera pas dans les solutions complètes héritant de cette solution partielle.
- On peut donc déclencher un retour arrière dès qu'une contrainte est violée.

Retours arrière

Algorithme 2 : FouilleEnProfondeurAvecRetoursArrière($\text{dom}(X_1), \ldots, \text{dom}(X_n)$)

répéter
  $Candidats \leftarrow \{X_i \mid |\text{dom}(X_i)| > 1\}$
  $SolutionPartielle \leftarrow \{X_i \mid |\text{dom}(X_i)| = 1\}$
  pour $C_j \in \mathcal{C}$ **faire**
    si *Portée*$(C_j) \subseteq SolutionPartielle$ *et* C_j *n'est pas satisfaite* **alors**
      retourner $\emptyset$
  si $Candidats = \emptyset$ **alors**
    retourner $\text{dom}(X_1), \ldots, \text{dom}(X_n)$
  Choisir une variable $X_i \in Candidats$
  Choisir une valeur $v \in \text{dom}(X_i)$
  $Solution \leftarrow$ FouilleEnProfondeurAvecRetoursArrière($\text{dom}(X_1), \ldots, \text{dom}(X_{i-1}), \{v\}, \text{dom}(X_{i+1}), \ldots, \text{dom}(X_n)$)
  si $Solution = \emptyset$ **alors**
    $\text{dom}(X_i) \leftarrow \text{dom}(X_i) \setminus \{v\}$
jusqu'à $Solution \neq \emptyset$
retourner $Solution$

Propriétés de la fouille avec retours arrière

- L'ensemble des noeuds explorés dans la fouille avec retours arrière est un sous-ensemble des noeuds explorés avec une fouille en profondeur sans retour arrière.
- L'élimination d'une branche en haut de l'arbre de recherche peut mener à un gain en temps de calcul d'une durée exponentielle.

Propriétés de la fouille avec retours arrière

- Certains problèmes qui auraient été résolus en temps exponentiel avec une fouille en profondeur sans retour arrière peuvent maintenant être résolus en temps polynomial.
- Nous aurons la chance de voir certains de ces problèmes plus tard dans le cours.

Vérification anticipée

- La vérification anticipée (*forward checking*) permet de réduire davantage le nombre de noeuds visités lors du processus de recherche.
- Cette technique consiste à détecter, pour chaque contrainte, si toutes les variables de la portée ont été instanciées à l'exception d'une variable.
- On itère alors sur chacune des valeurs du domaine de la variable non instanciée et on retire les valeurs qui ne satisfont pas la contrainte.

Exemple

- Nous avons la contrainte A + B < C avec les domaines suivants:
  - dom(A) = {1}, dom(B) = {1, 3, 5, 7}, dom(C) = {5}
- On teste la contrainte avec toutes les valeurs dans le domaine de B.

1 + 1 < 5	VRAI
1 + 3 < 5	VRAI
1 + 5 < 5	FAUX
1 + 7 < 5	FAUX

- Le domaine de B est filtré et devient dom(B) = {1, 3}

Algorithme 3 : FouilleEnProfondeurAvecVerificationsAnticipées(dom$(X_1), \ldots,$ dom(X_n))

répéter
  $VariablesDeLaSolutionPartielle \leftarrow \{X_i \mid |\text{dom}(X_i)| = 1\}$
  $ContraintesAVérifier \leftarrow \{C_j \mid |\text{Portée}(C_j) \setminus VariablesDeLaSolutionPartielle| = 1\}$
  tant que $ContraintesAVérifier \neq \emptyset$ **faire**
    Choisir une contrainte C_j dans $ContraintesAVérifier$
    Soit X_k la variable dans Portée(C_j) qui n'est pas fixée ou une variable quelconque dans Portée(C_j) si elles sont toutes fixées.
    pour $v \in dom(X_k)$ **faire**
      si *la solution partielle augmentée de* $X_k = v$ *ne satisfait pas la contrainte* C_j **alors**
        dom$(X_k) \leftarrow$ dom$(X_k) \setminus \{v\}$
    si $dom(X_k) = \emptyset$ **alors retourner** $\emptyset$
    si $|dom(X_k)| = 1$ **alors**
      $VariablesDeLaSolutionPartielle \leftarrow VariablesDeLaSolutionPartielle \cup \{X_k\}$
      $ContraintesAVérifier \leftarrow ContraintesAVérifier \cup \{C_j \mid X_k \in \text{Portée}(C_j) \wedge |\text{Portée}(C_j) \setminus VariablesDeLaSolutionPartielle| = 1\}$
      $ContraintesAVérifier \leftarrow ContraintesAVérifier \setminus \{C_j\}$
  $Candidats \leftarrow \{X_i \mid |\text{dom}(X_i)| > 1\}$
  si $Candidats = \emptyset$ **alors**
    retourner dom$(X_1), \ldots,$ dom(X_n)
  Choisir une variable $X_i \in Candidats$
  Choisir une valeur $v \in$ dom(X_i)
  $Solution \leftarrow$
  FouilleEnProfondeurAvecVerificationsAnticipées(dom$(X_1), \ldots,$ dom$(X_{i-1}), \{v\},$ dom$(X_{i+1}), \ldots,$ dom(X_n)
  si $Solution = \emptyset$ **alors**
    dom$(X_i) \leftarrow$ dom$(X_i) \setminus \{v\}$
jusqu'à $Solution \neq \emptyset$
retourner $Solution$

Exercice

- Considérez le problème suivant. Une fois que le solveur a branché sur l'affectation x_1 = vrai, quel est l'état des domaines après l'application de la vérification anticipée?

$$\neg x_1 \vee \neg x_2 \vee x_3$$

$$\neg x_1 \vee x_2$$

$$x_1, x_2, x_3 \in \{vrai, faux\}$$

Solution de l'exercice

- Seule la deuxième contrainte est à vérifier.
- Après vérification de la deuxième contrainte, on détecte que X_2 ne peut pas être fausse donc X_2 = vrai.
- Puisqu'il ne reste qu'une valeur dans le domaine de X_2, la variable X_2 fait maintenant partie de la solution partielle.
- La première contrainte est ajoutée aux contraintes à vérifier.
- On détecte que X_3 ne peut pas être fausse donc X_3 = vrai.
- Le problème est résolu.

Propriétés de la vérification anticipée

- La fouille avec vérification anticipée explore un sous-ensemble de noeuds explorés par la fouille avec retours arrière.

- Il existe des problèmes pour lesquels le solveur évite tous les retours arrière grâce à la vérification anticipée. Peu importe le choix de variable et de valeur, le solveur plonge dans l'arbre de recherche et atteint dès le premier coup une solution du problème.

Propriétés de la vérification anticipée

- La vérification anticipée est spécialement efficace lorsque la cardinalité des domaines est petite.
- Plus les domaines sont petits, plus il y a de chances qu'une variable soit intégrée à la solution partielle après la vérification. La vérification anticipée fonctionne donc très bien pour les variables booléennes.
- Cette recherche nécessite pour chaque contrainte un algorithme qui vérifie si la contrainte est satisfaite ou non.

Le filtrage des domaines

- Peut-on pousser plus loin cette idée de réduire le domaine des variables afin d'éviter des retours arrière lors de la fouille dans l'arbre de recherche?

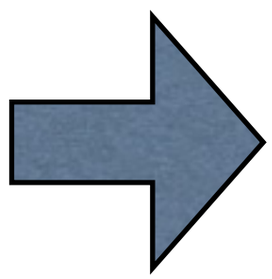
- Voici un problème très simple. Comment pourrait-on retirer des éléments des domaines afin d'éviter des retours arrière?

$$X_1 < X_2$$
$$\mathrm{dom}(X_1) = \{2, 3, 4, 5, 6\}$$
$$\mathrm{dom}(X_2) = \{1, 2, 3, 4\}$$

Le filtrage des domaines

- Il n'existe aucune solution avec $X_1 = 4$, $X_1 = 5$ ou $X_1 = 6$. Ces valeurs peuvent donc être retirées du domaine de X_1.
- Il n'existe aucune solution avec $X_2 = 1$ ou $X_2 = 2$. Ces valeurs peuvent elles aussi être retirées du domaine de X_2.

$$X_1 < X_2$$
$$\mathrm{dom}(X_1) = \{2, 3, 4, 5, 6\}$$
$$\mathrm{dom}(X_2) = \{1, 2, 3, 4\}$$



$$X_1 < X_2$$
$$\mathrm{dom}(X_1) = \{2, 3\}$$
$$\mathrm{dom}(X_2) = \{3, 4\}$$

Algorithme de filtrage

- Contrairement à la vérification anticipée qui ne demande qu'un **algorithme de vérification** testant si une solution satisfait une contrainte, le filtrage des domaines requiert un **algorithme de filtrage** plus complexe.

- L'algorithme de filtrage prend en entrée les domaines des variables dans la portée de la contrainte et modifie ces domaines pour retirer les valeurs qui ne mènent pas à une solution.

- Il faut un algorithme de filtrage par contrainte.

Algorithme de filtrage pour la contrainte <

- À l'exemple précédent, intuitivement, nous avons appliqué l'algorithme de filtrage de la contrainte <

Algorithme 4 : FiltrePlusPetitQue($\mathrm{dom}(X_1), \mathrm{dom}(X_2)$)

// Filtre les domaines de X_1 et X_2 selon la contrainte $X_1 < X_2$.

pour $v \in dom(X_1)$ **faire**
  si $v \geq \max(dom(X_2))$ **alors**
    $\mathrm{dom}(X_1) \leftarrow \mathrm{dom}(X_1) \setminus \{v\}$

pour $v \in dom(X_2)$ **faire**
  si $v \leq \min(dom(X_1))$ **alors**
    $\mathrm{dom}(X_2) \leftarrow \mathrm{dom}(X_2) \setminus \{v\}$

Support

- Afin de caractériser les algorithmes de filtrage, nous allons définir la notion de *support*.
- Un support est une affectation des variables qui satisfait une contrainte.
- Soit $C(X_1, ..., X_n)$ une contrainte.
  - Un **support de domaine** est un tuple *t* tel que $t[i] \in dom(X_i)$ et que la solution $\{X_i = t[i]\}_{i=1..n}$ satisfait la contrainte C.
  - Un **support d'intervalle** est un tuple *t* tel que $\min(dom(X_i)) \leq t[i] \leq \max(dom(X_i))$ et que la solution $\{X_i = t[i]\}_{i=1..n}$ satisfait la contrainte C.

Exemples de supports

- Considérez la contrainte et les domaines suivants.

$$X_1 + X_2 = X_3$$

$$\text{dom}(X_1) = \{1, 3\}, \text{dom}(X_2) = \{2, 5\}, \text{dom}(X_3) = \{3, 6\}$$

- Le tuple [1, 5, 6] est un support de domaine.
  - Nous avons 1 + 5 = 6
  - Chaque valeur du tuple est dans le domaine de sa variable respective.
- Le tuple [3, 2, 5] est un support d'intervalle
  - Nous avons 3 + 2 = 5
  - Chaque valeur est contenue dans le plus petit intervalle qui contient le domaine.

Propriétés des supports

- Un support de domaine est nécessairement un support d'intervalle.
  - Démonstration: Pour toute valeur $v \in dom(X)$ nous avons $min(dom(X)) \leq v \leq max(dom(X))$
- Un support de domaine garantit l'existence d'une solution pour la contrainte ce qui n'est pas le cas pour un support d'intervalle.
- On dit qu'il existe un support t pour la valeur $v \in dom(X_i)$ si t est un support et que $t[i] = v$.

Cohérence

- Un algorithme de filtrage retire des valeurs d'un domaine pour lesquelles **il n'existe pas de support**.
- Lorsque l'algorithme de filtrage a terminé de filtrer les domaines des variables sujettes à une contrainte, on dit que la contrainte est cohérente.
- Puisqu'il existe plusieurs types de support, il existe plusieurs niveaux de cohérence.

Niveaux de cohérence

- Une contrainte est **cohérente de domaine** si
  - pour toute variable X dans la portée de la contrainte et pour toute valeur v dans le domaine de X, il existe un *support de domaine* pour cette valeur.
- Une contrainte est **cohérente d'intervalle** si
  - pour toute variable X dans la portée de la contrainte et pour toute valeur v dans le domaine de X, il existe un *support d'intervalle* pour cette valeur.
- Une contrainte est **cohérente de bornes** si
  - pour toute variable X dans la portée de la contrainte, il existe un support d'intervalle pour la plus petite et la plus grande valeur du domaine de X.

Différence entre les cohérences

- Considérez la contrainte All-Different(X_1, X_2, X_3) qui est satisfaite lorsque toutes les variables prennent des valeurs distinctes.

$$dom(X_1) = \{1, 3\}$$
$$dom(X_2) = \{1, 3\}$$
$$dom(X_3) = \{1, 2, 3\}$$

- La valeur 1 dans le domaine de X_3 a le support d'intervalle [3, 2, 1], mais n'a pas de support de domaine.
- Cette contrainte n'est donc pas cohérente de domaine, mais elle est cohérente d'intervalle.

Différence entre les cohérences

- Considérez la contrainte All-Different(X_1, X_2, X_3) qui est satisfaite lorsque toutes les variables prennent des valeurs distinctes.

$$\text{dom}(X_1) = \{2, 3\}$$
$$\text{dom}(X_2) = \{2, 3\}$$
$$\text{dom}(X_3) = \{1, 2, 3, 4\}$$

- La valeur 2 dans le domaine de X_3 n'a pas de support d'intervalle. La contrainte n'est pas cohérente d'intervalle.
- Les deux supports d'intervalle [2, 3, 4] et [3, 2, 1] prouvent que la contrainte est cohérente de bornes.

Application d'une cohérence

- Appliquer une cohérence, c'est retirer des domaines les valeurs qui n'ont pas de support.
- Appliquer la cohérence de domaine, c'est retirer des domaines toutes les valeurs qui n'ont pas de support de domaine.
- Appliquer la cohérence d'intervalle, c'est retirer des domaines des variables toutes les valeurs qui n'ont pas de support d'intervalle.
- Appliquer la cohérence de bornes, c'est retirer les plus petits éléments des domaines jusqu'à ce qu'ils aient un support d'intervalle. On fait de même avec les plus grands éléments des domaines.

Exercices

1) Appliquez la cohérence de bornes à la contrainte suivante.

$$A + B = C$$
$$\mathrm{dom}(A) = \{1, 3\}$$
$$\mathrm{dom}(B) = \{1, 4\}$$
$$\mathrm{dom}(C) = \{2, 3, 4, 5, 9\}$$

2) Appliquez la cohérence de domaine à cette même contrainte.

Solutions

- Cohérence de bornes
  - dom(X1) = {1, 3}
  - dom(X2) = {1, 4}
  - dom(X3) = {2, 3, 4, 5}
- Cohérence de domaine
  - dom(X1) = {1, 3}
  - dom(X2) = {1, 4}
  - dom(X3) = {2, 4, 5}

Propriétés des cohérences

- Une contrainte qui est cohérente de domaine est nécessairement cohérente d'intervalle.
- Une contrainte qui est cohérente d'intervalle est nécessairement cohérente de bornes.
- Conséquemment, il est plus difficile d'appliquer la cohérence de domaine, que la cohérence d'intervalle, que la cohérence de bornes.
  - Ex: Il est NP-Difficile d'appliquer la cohérence de domaine à la contrainte Inter-Distance mais on peut lui appliquer la cohérence de bornes en temps $O(n^2)$.

Cohérence avec plusieurs contraintes

- Qu'en est-il lorsqu'un problème a plus d'une contrainte (ce qui est généralement le cas)?
- Nous pouvons d'abord appliquer la cohérence de domaine à la première contrainte puis la cohérence de domaine à la deuxième contrainte.
- Le filtrage effectué lors de l'application de la cohérence de domaine sur la deuxième contrainte peut avoir affecté la cohérence de la première contrainte.
- Il faut alors filtrer à nouveau la première contrainte.

Cohérence locale

- Après avoir itérativement filtré les contraintes d'un problème, on atteint un état où toutes les contraintes sont cohérentes, c'est ce qu'on appelle la **cohérence locale**.
- Un problème est localement cohérent lorsque toutes ses contraintes ont atteint le niveau de cohérence souhaité (généralement, la cohérence de domaine, la cohérence d'intervalle ou la cohérence de bornes).

Exemple de cohérence locale

- Appliquer la cohérence de bornes sur le problème de satisfaction de contraintes suivant jusqu'à ce que l'on obtienne la cohérence locale.

$$XY = 12$$
$$X + Y = 7$$
$$\mathrm{dom}(X) = [0, 12]$$
$$\mathrm{dom}(Y) = [0, 12]$$

Contrainte utilisée	dom(X)	dom(Y)
	[0, 12]	[0, 12]
XY = 12	[1, 12]	[1, 12]
X + Y = 7	[1, 6]	[1, 6]
XY = 12	[2, 6]	[2, 6]
X + Y = 7	[2, 5]	[2, 5]
XY = 12	[3, 4]	[3, 4]

Fouille avec filtrage

- Nous pouvons revoir l'algorithme de fouille avec retours arrière afin d'appliquer la cohérence locale avant chaque branchement.
- À chaque noeud de l'arbre de recherche, le solveur fera appel aux algorithmes de filtrage de ses contraintes pour éliminer des choix qui ne mènent pas à une solution.

Fouille avec filtrage

Algorithme 5 : FouilleEnProfondeurAvecFiltrage(dom$(X_1), \ldots,$ dom(X_n))

répéter
  Filtrer les domaines jusqu'à ce que l'on obtienne la cohérence locale.
  si *il existe un domaine qui a été vidé de ses valeurs* **alors**
    retourner $\emptyset$
  $Candidats \leftarrow \{X_i \mid |\text{dom}(X_i)| > 1\}$
  si $Candidats = \emptyset$ **alors**
    retourner dom$(X_1), \ldots,$ dom(X_n)
  Choisir une variable $X_i \in Candidats$
  Choisir une valeur $v \in$ dom(X_i)
  $Solution \leftarrow$ FouilleEnProfondeurAvecFiltrage(dom$(X_1), \ldots,$ dom$(X_{i-1}),$
    $\{v\},$ dom$(X_{i+1}), \ldots,$ dom$(X_n))$
  si $Solution = \emptyset$ **alors**
    dom$(X_i) \leftarrow$ dom$(X_i) \setminus \{v\}$
jusqu'à $Solution \neq \emptyset$
retourner $Solution$

Comment obtenir la cohérence locale

- Un algorithme de force brute pourrait tout simplement itérer sur toutes les contraintes du problème et exécuter leurs algorithmes de filtrage. Cette itération reprend du début jusqu'à ce qu'aucun algorithme de filtrage ne puisse modifier les domaines des variables.
- Cette solution a l'avantage d'être simple, mais elle a le potentiel d'appeler inutilement certains algorithmes de filtrage. En effet, certaines contraintes peuvent déjà être cohérentes.
- Afin de gagner en efficacité, il faut identifier les contraintes devant être filtrées.

La propagation des contraintes

- À chaque noeud de l'arbre de recherche, on applique la cohérence locale puis on instancie une variable. Soit X la variable instanciée.
- Parmi toutes les variables du problème, seule la variable X invalide la cohérence locale.
- Nous pouvons seulement filtrer les contraintes ayant X dans leur portée.
- Si, lors de ce filtrage, le domaine d'une variable Y est modifié, il faudra alors filtrer les contraintes ayant Y dans leur portée et ainsi de suite.
- Ce processus s'appelle la **propagation des contraintes**.

La propagation des contraintes

- La technique consiste à garder une liste de contraintes potentiellement incohérentes.
- Initialement, si la variable X vient d'être instanciée, on sait que toutes les contraintes ayant X dans leur portée sont potentiellement incohérentes.
- À chaque fois que le domaine d'une variable est modifié, il faut ajouter à la liste les contraintes ayant la variable modifiée dans leur portée.

Algorithme de propagation

Algorithme 6 : PropagationDesContraintes(X_i, $C_1, \ldots, C_m$)

Soit X_i la variable instanciée.
$S \leftarrow \{C_k \mid X_i \in \text{Portée}(C_k)\}$
tant que $S \neq \emptyset$ **faire**
    Choisir une contrainte C_k dans S
    Filtrer la contrainte C_k
    si *la contrainte est insatisfiable* **alors**
        retourner « Échec »
    pour chaque *variable* X_j *dont le domaine a été filtré* **faire**
        $S \leftarrow S \cup \{C_a \mid X_j \in \text{Portée}(C_a)\}$
    $S \leftarrow S \setminus \{C_k\}$
retourner « Cohérent »

Exemple

$$XY = Z$$
$$X < Y$$
$$X = 2W$$
$$\mathrm{dom}(W) = [0, 10]$$
$$\mathrm{dom}(X) = [0, 39]$$
$$\mathrm{dom}(Y) = [1, 40]$$
$$\mathrm{dom}(Z) = [24, 24]$$

Z est la variable instanciée

Contrainte	dom(W)	dom(X)	dom(Y)	S
	[0, 10]	[0, 39]	[1, 40]	{XY = Z}
XY = Z	[0, 10]	[1, 24]	[1, 24]	{X < Y, X = 2W}
X < Y	[0, 10]	[1, 23]	[2, 24]	{X = 2W, XY = Z}
X = 2W	[1, 10]	[2, 20]	[2, 24]	{XY = Z, X < Y}
XY = Z	[1, 10]	[2, 12]	[2, 12]	{X < Y, X = 2W}
X < Y	[1, 10]	[2, 11]	[3, 12]	{X = 2W, XY = Z}
X = 2W	[1, 5]	[2, 10]	[3, 12]	{XY = Z, X < Y}
XY = Z	[1, 5]	[2, 8]	[3, 12]	{X < Y, X = 2W}
X < Y	[1, 5]	[2, 8]	[3, 12]	{X = 2W}
X = 2W	[1, 4]	[2, 8]	[3, 12]	{}

Comment l'implanter?

- L'algorithme permet de choisir arbitrairement les contraintes à traiter.
- Tous les choix ne sont pas équivalents.
- Il faut trouver une heuristique qui mènera à un échec ou une cohérence locale le plus rapidement possible.
- On ordonne la file S en donnant priorité aux contraintes ayant le potentiel de filtrer la plus grande quantité de valeurs dans le plus court laps de temps possible.
- En pratique, on repousse au plus tard les contraintes ayant un algorithme de filtrage complexe et lent.

Apprentissage de clauses

- Celui qui oublie est condamné à répéter ses erreurs.
- Dans un arbre de recherche, il arrive que certaines combinaisons de variables soient essayées plusieurs fois en vain.
- Apprendre les raisons d'un échec nous permet de les éviter et même d'améliorer les heuristiques de choix de variables et de valeurs pendant la recherche.

Apprentissage de clauses

- Pendant la recherche, nous pouvons ajouter de nouvelles variables et contraintes au modèle.
- Ces ajouts ne changent en rien l'espace des solutions.
- Les algorithmes de filtrage de ces nouvelles contraintes permettront d'éviter les choix qui ont mené à un retour arrière.

Ajouts au modèle

- Durant la recherche, nous ajoutons des variables booléennes des formes $[\![X = v]\!]$ et $[\![X \leq v]\!]$.
- La relation $X \geq v$ est encodée avec $\neg[\![X \leq v - 1]\!]$.
- Ces variables booléennes peuvent encoder n'importe quel domaine d'une variable entière. Ex:

$$\mathrm{dom}(X) = \{1, 2, 4\} \iff \neg[\![X \leq 0]\!] \wedge \neg[\![X = 3]\!] \wedge [\![X \leq 4]\!]$$

- Les contraintes qui sont ajoutées pendant la recherche sont des clauses SAT, c'est-à-dire des disjonctions de variables ou de leur négation.

Contraintes implicites

- Pour une variable *X* avec domaine dom(X) = {l, …, u}, nous avons ces contraintes implicitement ajoutées au modèle.

$$[\![X \leq u]\!]$$

$$\neg[\![X \leq l-1]\!]$$

$$[\![X \leq v]\!] \implies [\![X \leq v+1]\!]$$

$$[\![X \leq v]\!] \wedge \neg[\![X = v]\!] \implies [\![X \leq v-1]\!]$$

$$\neg[\![X \leq v-1]\!] \wedge \neg[\![X = v]\!] \implies \neg[\![X \leq v]\!]$$

$$[\![X \leq v]\!] \wedge \neg[\![X \leq v-1]\!] \implies [\![X = v]\!]$$

- Note: les 2 dernières contraintes sont équivalentes.

Explications

- Lorsqu'un algorithme de filtrage procède à un filtrage, il doit l'expliquer à l'aide d'une implication logique.
- Ex: dom(X) = dom(Y) = dom(Z) = [1, 3], X + Y = Z

variable	domaine filtré	Explication
X	[1, 2]	$\neg[\![Y \leq 0]\!] \wedge [\![Z \leq 3]\!] \implies [\![X \leq 2]\!]$
Y	[1, 2]	$\neg[\![X \leq 0]\!] \wedge [\![Z \leq 3]\!] \implies [\![Y \leq 2]\!]$
Z	[2, 3]	$\neg[\![X \leq 0]\!] \wedge \neg[\![Y \leq 0]\!] \implies \neg[\![Z \leq 1]\!]$

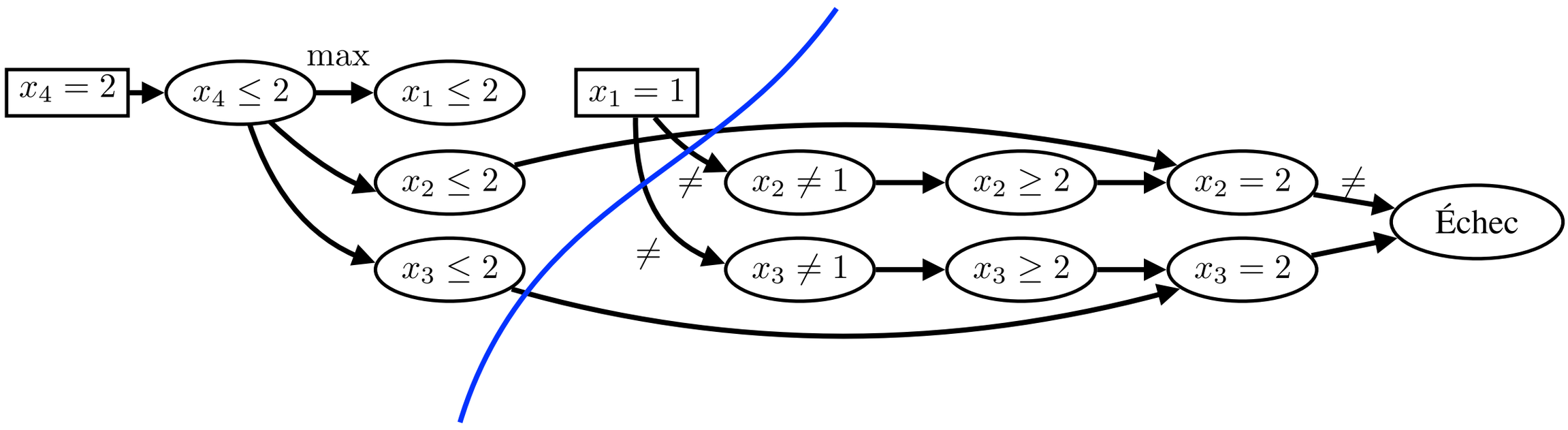
- Durant la propagation des contraintes, ces explications sont ajoutées à un graphe d'implications.

Apprentissage d'une clause

- Une coupe dans le graphe d'implication sépare les branchements (noeuds rectangulaires) du noeud d'échec.
- Les noeuds à l'origine des arcs coupés sont des conditions suffisantes pour causer un échec.
- On impose donc la disjonction de leurs négations.
- Un exemple à la page suivante suit où un solveur effectue les branchements $x_4 = 2$ et $x_1 = 1$.

Une trace

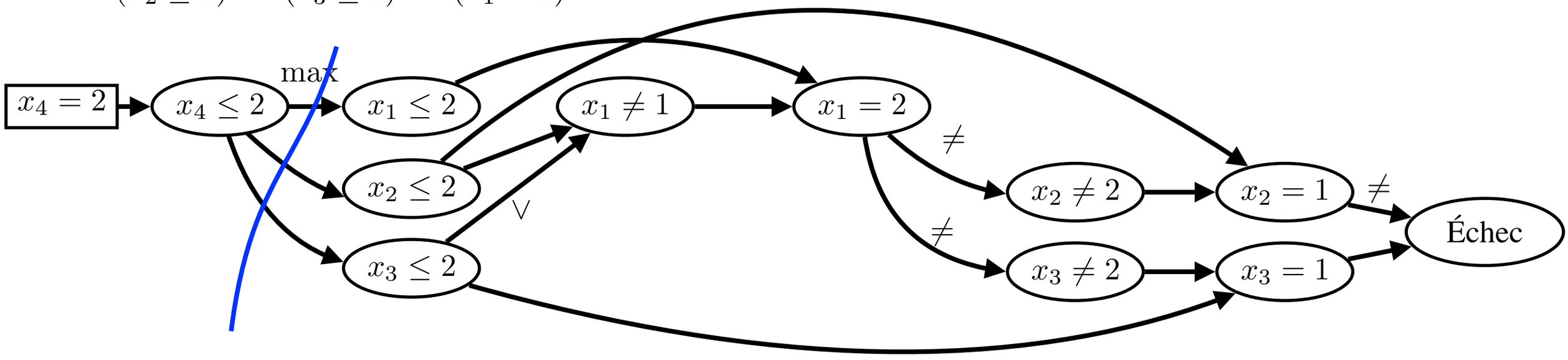
$$\mathrm{dom}(x_i) = \{1, 2, 3\} \qquad \forall i = 1..4$$
$$x_4 = \max([x_1, x_2, x_3])$$
$$x_1 \neq x_2$$
$$x_2 \neq x_3$$
$$x_1 \neq x_3$$



Clause apprise: $\neg(x_2 \leq 2) \vee \neg(x_3 \leq 2) \vee \neg(x_1 = 1)$

Une trace

$$\mathrm{dom}(x_i) = \{1,2,3\} \qquad \forall i = 1..4$$
$$x_4 = \max([x_1, x_2, x_3])$$
$$x_1 \neq x_2$$
$$x_2 \neq x_3$$
$$x_1 \neq x_3$$
$$\neg(x_2 \leq 2) \vee \neg(x_3 \leq 2) \vee \neg(x_1 = 1)$$



Clause apprise: $\neg(x_4 \leq 2)$

Observation

- Remarquez qu'avec les nouvelles clauses apprises, le solveur va filtrer la valeur 1 du domaine de x_4.
- En effet, brancher sur $x_4 = 1$ mène à un échec pour les mêmes raisons que $x_4 = 2$ mène à un échec.

La coupe 1UIP

- Certaines coupes génèrent de meilleures clauses que d'autres.
- La technique 1UIP (first unique implication point) offre les meilleures performances empiriques.
- Un *point unique d'implication* est un noeud par lequel passe tous les chemins reliant le dernier branchement au noeud *échec*.
- Le *premier point unique d'implication* est le point unique d'implication le plus près du noeud échec.
- On place la coupe immédiatement à droite du premier point unique d'implication.

Branch & Bound

- Lors de la résolution d'un problème d'optimisation, il ne suffit pas de retourner la première solution qui satisfait toutes les contraintes. Il faut trouver la solution qui optimise la fonction objectif.

- Considérons un problème dont la fonction objectif est de minimiser la variable *X*.

- Après avoir trouvé une solution où $X = v$, on ajoute au problème la contrainte $X < v$ et on poursuit l'exploration de l'arbre de recherche.

Branch & Bound

- Le filtrage de la variable *X* devient particulièrement important.
- La contrainte *X* < *v* filtre la borne supérieure du domaine de *X* et garantie de ne trouver que de meilleurs solutions.
- Les autres contraintes du problème filtrent la borne inférieure du domaine de *X*. Elles calculent donc une borne optimiste.
- Si la borne optimiste (inférieure) sur *X* est supérieure à la meilleure solution trouvée, alors le domaine de *X* devient vide et le retour arrière est déclenché.

Des cohérences plus fortes

- Une cohérence A est dite plus forte qu'une cohérence B si appliquer la cohérence A filtre un surensemble des valeurs filtrées par la cohérence B.
- Il existe des cohérences plus fortes que la cohérence locale.
- Nous verrons deux techniques qui permettent de filtrer davantage les domaines des variables.

La cohérence de singleton

- Le cohérence de singleton consiste à affecter temporairement une valeur à une variable. C'est l'équivalent d'un branchement dans un arbre de recherche.
- Une fois la variable fixée, on applique la cohérence locale.
- Deux conséquences peuvent s'ensuivre:
  - Il n'y a pas de cohérence locale (ex: le filtrage vide le domaine d'une variable);
  - Il y a une cohérence locale.
- Dans le premier cas, on retire la valeur du domaine de la variable.
- Dans le deuxième cas, on conserve la valeur dans le domaine de la variable et procède avec une autre paire (variable, valeur)

Exemple de cohérence de singleton

$$X_1 \neq X_2$$
$$X_1 \neq X_3$$
$$X_2 \neq X_3$$

$$\mathrm{dom}(X_1) = \{1, 2, 3\}$$
$$\mathrm{dom}(X_2) = \{1, 2\}$$
$$\mathrm{dom}(X_3) = \{1, 2\}$$

- Appliquer la cohérence de singleton retire les valeurs 1 et 2 du domaine de X_1.

Note à propos de la cohérence de singleton

- Notez que si l'application de la cohérence de singleton retire une valeur du domaine, il se peut que les affectations précédemment testées ne soient plus cohérentes. En effet, la valeur retirée du domaine pourrait être l'unique support d'une valeur déjà testée.
- Même si on peut obtenir un meilleur filtrage en revisitant les valeurs déjà testées, cette technique est si lente qu'on se contente généralement de tester chaque valeur une seule fois.

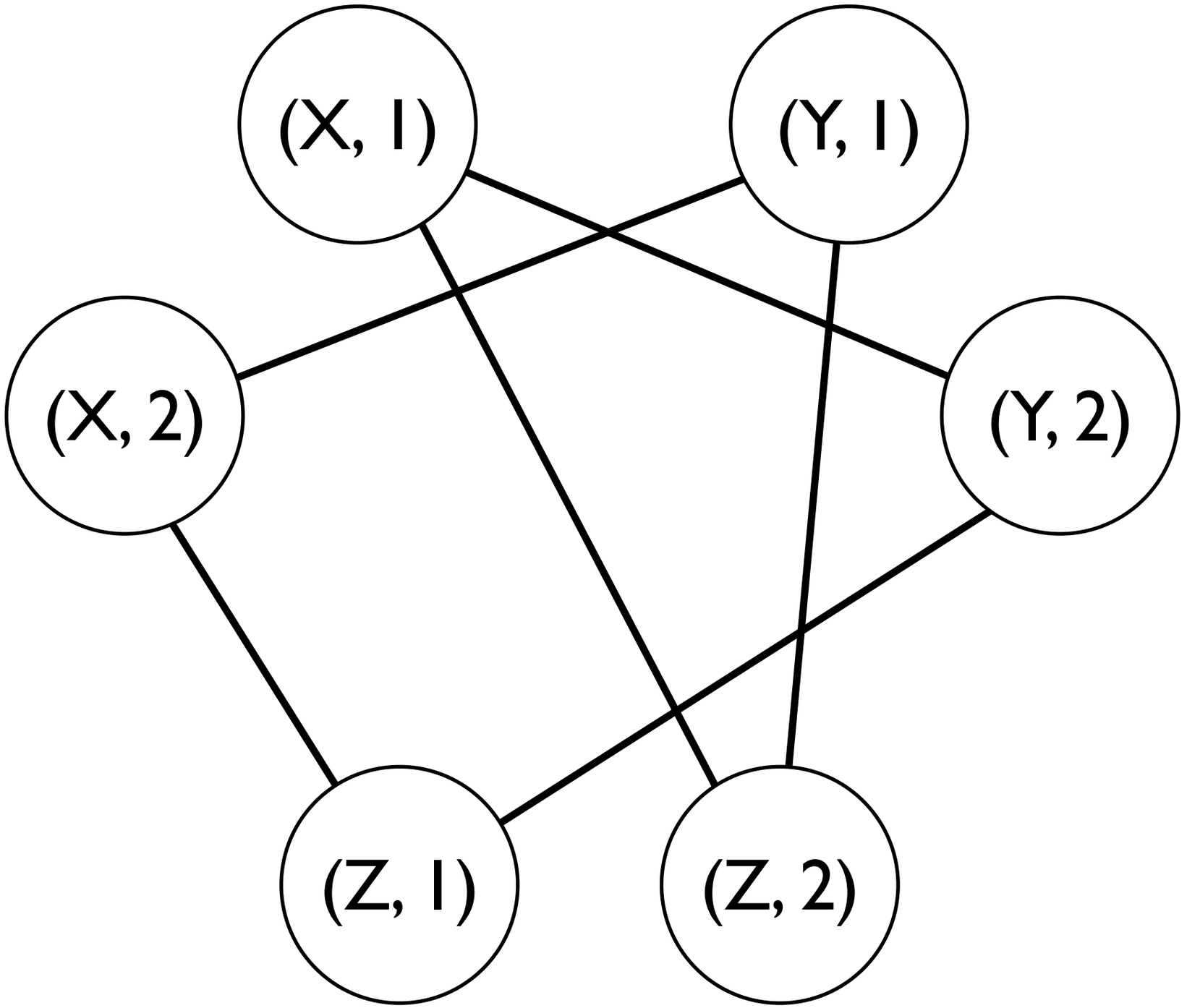
La cohérence de chemin

- La cohérence de chemin est une technique de filtrage pour les problèmes de satisfaction de contraintes binaires, c'est-à-dire les problèmes de satisfaction où toutes les contraintes n'ont que deux variables dans leur portée.

- Elle consiste à créer un graphe où chaque paire (X, v) telle que v $\in$ dom(X) est un noeud.

- On ajoute une arête entre le noeud (X, v) et (Y, u) s'il n'existe aucune contrainte binaire dont la portée est {X, Y} qui empêche la solution partielle {X = v, Y = u}.

Exemple



$$X \neq Y$$
$$X \neq Z$$
$$Y \neq Z$$
$$\mathrm{dom}(X) = \{1, 2\}$$
$$\mathrm{dom}(Y) = \{1, 2\}$$
$$\mathrm{dom}(Z) = \{1, 2\}$$

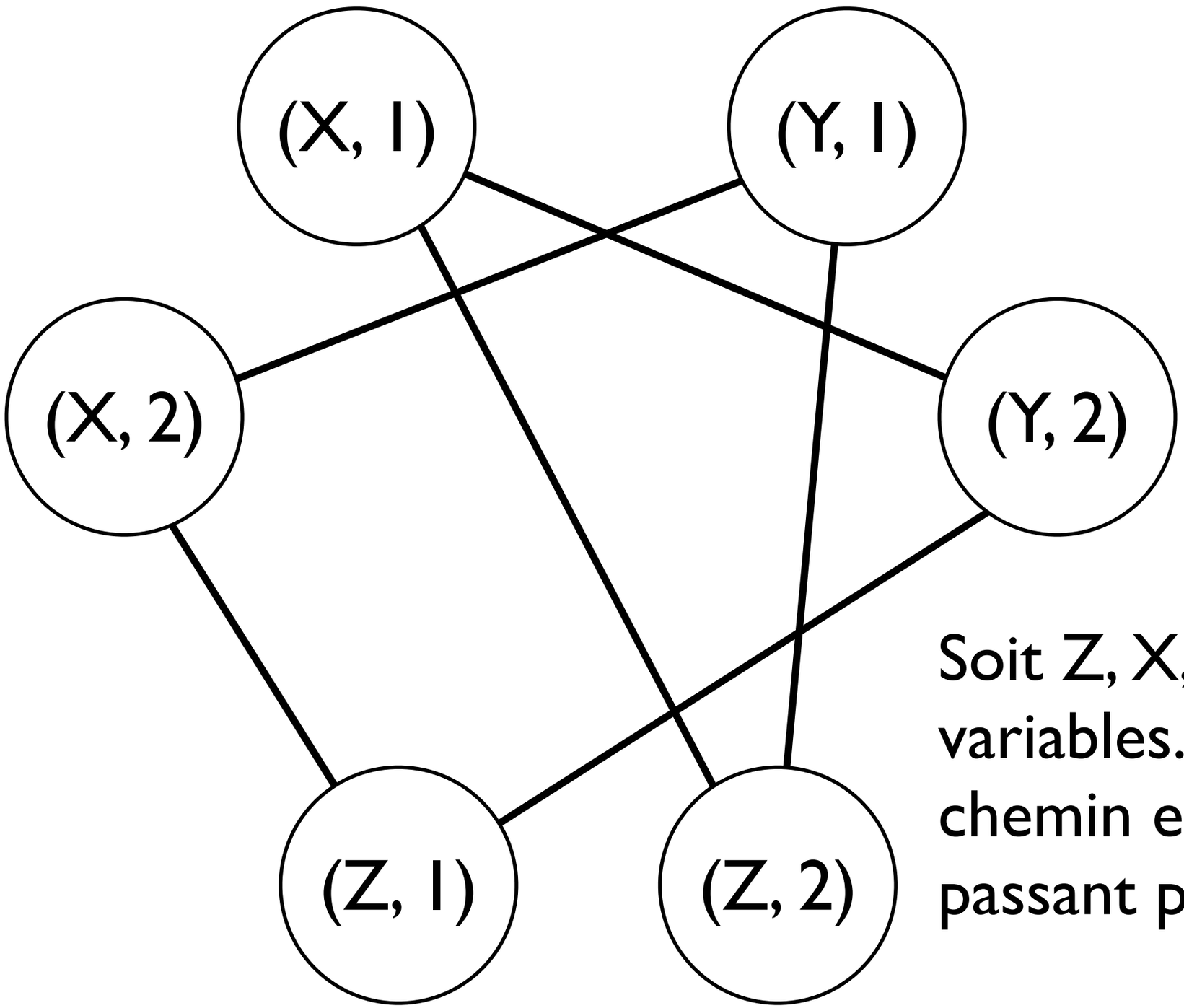
La cohérence de chemin

- Un problème est **cohérent de chemin** si pour chaque arête $((X_i, v_i), (X_j, v_j))$ du graphe et pour toute séquence de $k \leq n$ variables $X_{p[1]}, ..., X_{p[k]}$ où $p[1] = i$ et $p[k] = j$, il existe des valeurs $v_{p[1]}, ..., v_{p[k]}$ telles que $(X_{p[1]}, v_{p[1]}), (X_{p[2]}, v_{p[2]}), ..., (X_{p[k]}, v_{p[k]})$ est un chemin dans le graphe.

- Pour appliquer la cohérence de chemin, on considère une arête du graphe $((X_i, v_i), (X_j, v_j))$ et toutes les permutations de k variables $X_{p[1]}, ..., X_{p[k]}$ commençant par X_i et finissant par X_j et ce, pour toute valeur de $k \leq n$. S'il n'y a pas de chemin entre le noeud (X_i, v_i) et le noeud (X_j, v_j) passant par les variables $X_{p[2]}, ..., X_{p[k-1]}$, alors on retire l'arête $((X_i, v_i), (X_j, v_j))$ du graphe.

- Notez que l'on retire des arêtes du graphe et non pas des valeurs des domaines. Si toutes les arêtes adjacentes à un noeud (X_i, v) sont retirées, on retire alors la valeur v du domaine de X_i.

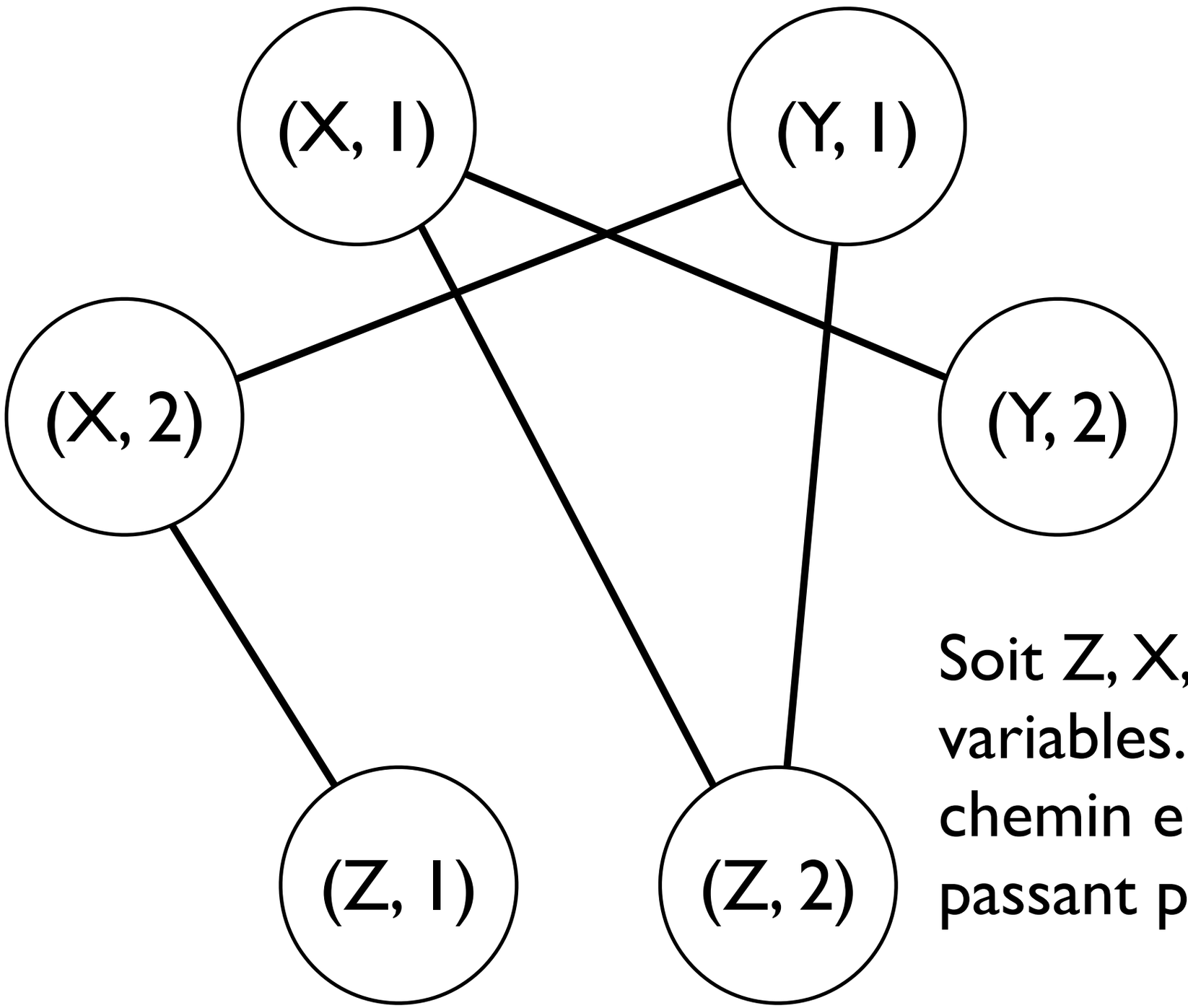
Exemple



$$X \neq Y$$
$$X \neq Z$$
$$Y \neq Z$$
$$\mathrm{dom}(X) = \{1, 2\}$$
$$\mathrm{dom}(Y) = \{1, 2\}$$
$$\mathrm{dom}(Z) = \{1, 2\}$$

Soit Z, X, Y une séquence de variables. Il n'existe pas de chemin entre (Z, 1) et (Y, 2) passant par cette séquence.

Exemple

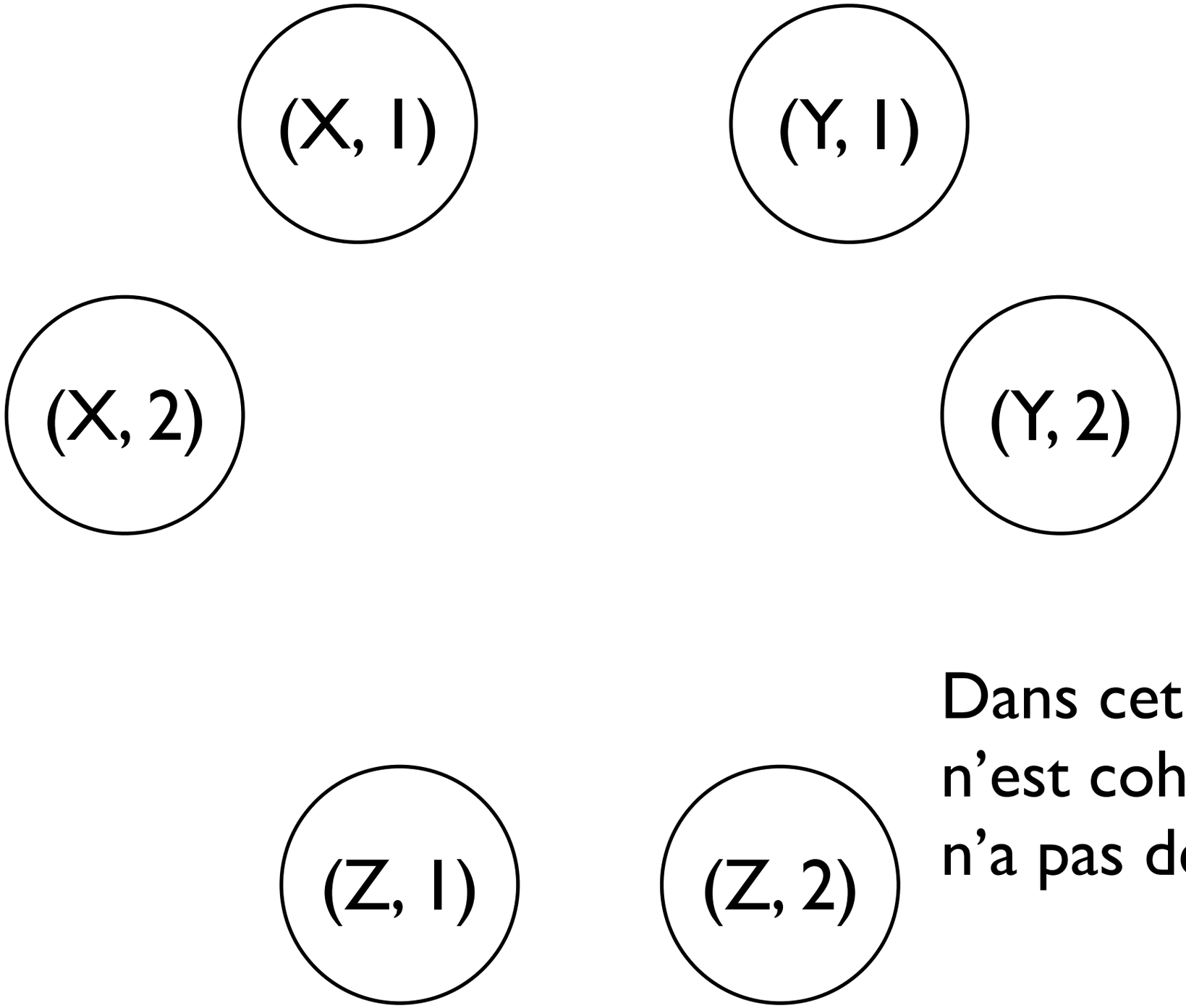


$$X \neq Y$$
$$X \neq Z$$
$$Y \neq Z$$
$$\mathrm{dom}(X) = \{1, 2\}$$
$$\mathrm{dom}(Y) = \{1, 2\}$$
$$\mathrm{dom}(Z) = \{1, 2\}$$

Soit Z, X, Y une séquence de variables. Il n'existe pas de chemin entre (Z, 1) et (Y, 2) passant par cette séquence.

Exemple

$$X \neq Y$$
$$X \neq Z$$
$$Y \neq Z$$
$$\mathrm{dom}(X) = \{1, 2\}$$
$$\mathrm{dom}(Y) = \{1, 2\}$$
$$\mathrm{dom}(Z) = \{1, 2\}$$



Dans cet exemple, aucun arc n'est cohérent et le problème n'a pas de solution.

Efficacité de la méthode

- Appliquée telle que décrite, la cohérence de chemin est inefficace.
- En effet, il existe n! façons de permuter n variables donc la complexité sera factorielle seulement pour tester les chemins de longueur n.
- Pour la majorité des problèmes, il serait plus rapide de trouver une solution au problème que d'appliquer la cohérence de chemin en utilisant la technique décrite précédemment.
- Nous proposons une cohérence plus simple à appliquer.

Cohérence de chemin de longueur 2

- Un problème est **cohérent de chemin de longueur 2** si pour chaque arête $((X_i, v_i), (X_k, v_k))$ et pour chaque variable X_j, il existe un chemin de longueur deux $(X_i, v_i), (X_j, v_j), (X_k, v_k)$.
- Pour appliquer la cohérence de chemin de longueur 2, on considère un arc du graphe $((X_i, v), (X_k, u))$ et toutes les variables X_j. S'il n'y a pas de chemin entre le noeud (X_i, v) et le noeud (X_k, u) passant par la variable X_j, alors on retire l'arc $((X_i, v), (X_j, u))$ du graphe.
- Cette technique ne considère donc que les triplets de variable plutôt que les permutations. Il y a $\binom{n}{3}$ triplets possibles plutôt que n! permutations de variables.

Comparaison entre les deux cohérences de chemin

- Question: Laquelle des deux cohérences est la plus forte? La cohérence de chemin ou la cohérence de chemin de longueur 2?
- Réponse: Elles sont équivalentes!

Chemin est aussi fort que Chemin 2

- Supposons que le problème est cohérent de chemin. Pour chaque arête $((X_i, v_i), (X_j, v_j))$, pour chaque $k \leq n$, pour chaque permutation de k variables, il existe un chemin allant de (X_i, v_i) à (X_j, v_j) en passant par toutes les variables de la permutation.
- Si l'énoncé est vrai pour toute valeur de $k \leq n$, il est vrai pour $k = 2$.
- Donc, si le problème est cohérent de chemin, il est cohérent de chemin de longueur 2.

Chemin 2 est aussi fort que Chemin n

- Soit $((X_{p[1]}, v_{p[1]}), (X_{p[n]}, v_{p[n]}))$ une arête du graphe et soit p une permutation quelconque.
- Hypothèse d'induction: Supposons que le problème est cohérent de chemin de longueur 2 et qu'il existe un chemin $(X_{p[1]}, v_{p[1]}), (X_{p[2]}, v_{p[2]}), ..., (X_{p[i]}, v_{p[i]}),(X_{p[n]}, v_{p[n]})$
- Puisqu'il existe une arête $((X_{p[i]}, v_{p[i]}), (X_{p[n]}, v_{p[n]}))$, il existe une valeur $v_{p[i+1]}$ telle que $(X_{p[i]}, v_{p[i]})$, $(X_{p[i+1]}, v_{p[i+1]}), (X_{p[n]}, v_{p[n]})$ est un chemin.
- Conséquemment, il existe un chemin $(X_{p[1]}, v_{p[1]}), (X_{p[2]}, v_{p[2]}), ..., (X_{p[i]}, v_{p[i]}), (X_{p[i+1]}, v_{p[i+1]}),(X_{p[n]}, v_{p[n]})$

Complexité

- Un chemin de longueur 2 est composé de 3 noeuds.
- Puisqu'il existe $O(n^3)$ façons de choisir trois variables, tester tous les chemins de longueur 2 peut se faire en temps polynomial.
- Conséquemment, appliquer la cohérence de chemin se fait en temps polynomial.
- Les meilleurs algorithmes appliquent la cohérence de chemin en temps $O(n^3d^3)$ où n est le nombre de variables et d est la cardinalité du plus grand domaine.

Référence: C.C. Han and C.H. Lee. Comments on Mohr and Henderson's path consistency algorithm. Artificial Intelligence, 36:125–130, 1988.

Conclusion

- Les solveurs de contraintes utilisent une fouille dans un arbre de recherche pour énumérer toutes les solutions candidates possibles.
- Un arbre de recherche peut prendre plusieurs formes possibles selon l'heuristique utilisée.
- La vérification anticipée requiert un algorithme de vérification pour chaque contrainte et permet de déclencher des retours arrière plus haut dans l'arbre de recherche.
- Le filtrage requiert un algorithme de filtrage pour chaque contrainte. Il existe différents niveaux de cohérence pour une contrainte. Les trois principaux sont les cohérences de domaine, d'intervalle et de bornes.

Conclusion

- L'apprentissage de clause requiert des algorithmes de filtrage capables d'expliquer le filtrage effectué.
- Un problème est localement cohérent lorsque toutes ses contraintes sont cohérentes.
- Le cohérence de singleton est plus forte que la cohérence locale.
- La cohérence de chemin est aussi plus forte que la cohérence locale.

Références bibliographiques

- Définition d'un problème de satisfaction de contraintes
  - Section 2.2.1 dans le Handbook of Constraint Programming (HCP)
- Fouille avec filtrage
  - Section 4.3 sauf 4.3.2 dans le HCP
- Définition des supports et des cohérences
  - Section 2 dans A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. *A fast and simple algorithm for bounds consistency of the alldifferent constraint.* In Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 03), pages 245-250, 2003.

Références bibliographiques

- Génération de clauses
  - Niklas Eén and Niklas Sörensson, *An Extensible SAT-Solver, Theory and Applications of Satisfiability Testing (SAT 2003), pages 502-518, 2003.*
  - Olga Ohrimenko, Peter J. Stuckey, and Michael Codish, *Propagation via Lazy Clause Generation,* Constraints 14(3), pages 357-391, 2009.
- Cohérence de singleton
  - Section 3.5.3 dans le HCP
- Cohérence de chemin
  - Section 3.4.1 dans le HCP

Traductions

Français	Anglais
Portée (d'une contrainte)	Scope
Fouille en profondeur	Depth-First-Search (DFS)
Fouille avec retours-arrière	Backtracking Search
Vérification anticipée	Forward Checking
Cohérence de domaine	Domain consistency / Generalized Arc Consistency(GAC)
Cohérence d'intervalle	Range consistency (RC)
Cohérence de bornes	Bounds consistency (BC)
Cohérence de chemin	Path consistency
Algorithme de filtrage	Filtering algorithm / Propagator