Combinatoire, Complexité et Graphes

IA-CCG

Samba Ndojh NDIAYE
Laboratoire d'InfoRmatique en Image et Systèmes d'information
LIRIS UMR 5205 CNRS/Université Claude Bernard Lyon 1
samba-ndojh.ndiaye@univ-lyon1.fr
perso.liris.cnrs.fr/samba-ndojh.ndiaye/enseignements.html

Sommaire

Séances

CM et TP

- CM : 12h
- TP : 8h (Présentations des projets)

Examen

- Examen : 23/01 à 14h
- Correction examen année dernière : 21/11

Projet

Résolution d'un problème avec un solveur de contraintes

- Présentation 1 : 15mn (+5mn questions) –> 03/10
- Présentation 2 : 15mn (+5mn questions) –> 28/11
- Groupes : 2 étudiants

Solveurs de contraintes

Résolution d'un problème avec un solveur de contraintes

- Choisir un problème NP-Complet ou NP-Difficile
- Disposer d'un benchmark d'instances pour ce problème (formats : minizinc, flatzinc, xcsp...)
- Choisir un (ou 2) solveur de contraintes paramétrable (au moins 2 paramètres) : Choco, Gecode, Glucose, OR-Tools, OSCAR/CBLS...
- Définir un modèle (voire plusieurs) pour le problème

Méthodes complètes

- Définir une méthode efficace (voire plusieurs) de résolution complète, en plus de la stratégie par défaut du solveur

Méthodes incomplètes

- Définir une méthode efficace (voire plusieurs) de résolution incomplète, en plus de la stratégie par défaut du solveur
- ou programmer directement une méthode incomplète (sans solveur)

Projet

Présentation 1

Problème, solveur, modèles, benchmark...

Présentation 2

Rappels (Problème, solveur, modèles), méthodes, benchmark, résultats, analyse...

Choix avant le 26/09

- Par mél : binôme + problème + solveur
- Pas plus de 2 binômes sur un même problème : FIFO

Évaluation

Projet

- Rendu : Présentation 2 + Code –> note
- Présentation 1 : non évaluée

Examen

- Questions sur le cours
- Questions portant sur votre projet
- Aucun document autorisé

Note Finale : Coefficients

- Examen : 60%
- Projet : 40%

Supports de cours

Supports de Christine Solnon

- http://perso.citi.insa-lyon.fr/csolnon/
- Cours en Master recherche : Résolution de problèmes combinatoires
- Diapositives
- Support de cours

Un Condensé + Détails sur les approches complètes

- perso.liris.cnrs.fr/samba-ndojh.ndiaye/enseignements.html

Sommaire

Sommaire

Complexité : Rappels

Complexité d'un algorithme

- Estimation des ressources nécessaires à l'exécution
  - Complexité en temps = estimation du nombre d'instructions
  - Complexité en espace = estimation de l'espace mémoire

  $\rightsquigarrow$ comparer 2 algorithmes / la taille n des données en entrée
- Ordre de grandeur $\mathcal{O}$

  $\mathcal{O}(f(n)) \rightsquigarrow \exists c,\ n_0\ tq\ \forall n > n_0, nb\ instructions < c.f(n)$
  - $\mathcal{O}(log(n))$: logarithmique
  - $\mathcal{O}(n)$: linéaire
  - $\mathcal{O}(n^k)$: polynomial
  - $\mathcal{O}(k^n)$: exponentiel

Pour en savoir plus

Introduction à l'algorithmique de Cormen, Leiserson et Rivest

Complexité de quelques algorithmes

Rechercher un élément dans un tableau de n éléments

- Algorithme de recherche séquentielle : $\mathcal{O}(n)$
- Algorithme de recherche dichotomique / tableau trié : $\mathcal{O}(log(n))$

4	8	2	1	5

1	2	4	5	8

Complexité de quelques algorithmes

Trier un tableau de n éléments

- Algorithme de tri par sélection : $\mathcal{O}(n^2)$
- Algorithme de tri par insertion :
  $\rightsquigarrow$ dans le meilleur des cas (tableau déjà trié) : $\mathcal{O}(n)$
  $\rightsquigarrow$ dans le pire des cas (tableau trié à l'envers) : $\mathcal{O}(n^2)$
- Algorithme de tri rapide (quicksort) :
  $\rightsquigarrow$ dans le meilleur des cas (pivot = médiane) : $\mathcal{O}(n.log(n))$
  $\rightsquigarrow$ dans le pire des cas (pivot = elt max ou min) : $\mathcal{O}(n^2)$
- Algorithme de tri par tas : $\mathcal{O}(n.log(n))$

4	8	2	1	5

Complexité de quelques algorithmes

Maximiser une fonction linéaire sous des contraintes linéaires

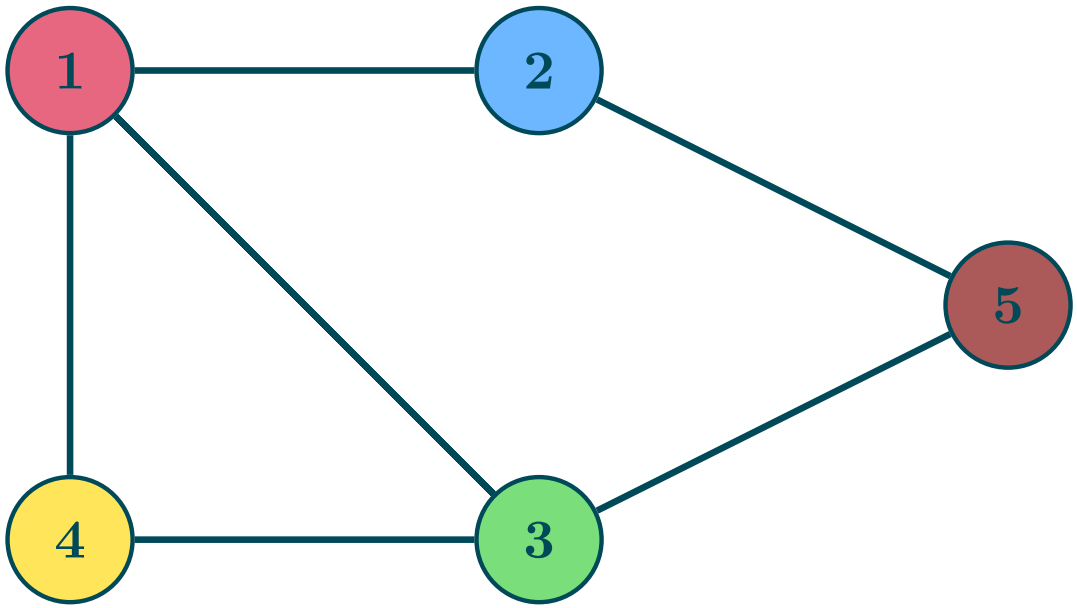
- Algorithme du simplexe : $\mathcal{O}(2^n)$ dans le pire des cas...
  faiblement polynomial en pratique !
- Algorithme du point intérieur : polynomial

maximiser $f = 3x + 4y - 2z$
tel que $2x + z \geq 10$
$x - 3y \leq 30$

Complexité de quelques algorithmes

Chercher un cycle hamiltonien dans un graphe

- Algorithme énumérant toutes les permutations des sommets : $\mathcal{O}(n!)$



1	2	3	4	5
2	1	3	4	5
3	2	1	4	5
4	2	3	1	5
5	2	3	4	1

...

Sommaire

Complexité d'un problème

Complexité d'un problème X

- Complexité du meilleur algorithme résolvant X...
- ... pour l'instance la plus difficile de X
  Une instance de X = une valuation des données en entrée de X

Classes de complexité

- Définies par rapport à la machine de Turing
- Pour les problèmes de décision seulement
  $\rightsquigarrow$ réponse = oui ou non
  Exemple : 4 appartient-il au tableau $[1, 2, 4, 6, 12]$?
- Principales classes : P, NP, NP-complet, NP-difficile, Indécidable

Complexité d'un problème

Complexité d'un problème X

- Complexité du meilleur algorithme résolvant X...
- ... pour l'instance la plus difficile de X
  Une instance de X = une valuation des données en entrée de X

Classes de complexité

- Définies par rapport à la machine de Turing
- Pour les problèmes de décision seulement
  $\rightsquigarrow$ réponse = oui ou non
  Exemple : 4 appartient-il au tableau $[1, 2, 4, 6, 12]$?
- Principales classes : P, NP, NP-complet, NP-difficile, Indécidable

Classes de complexité

La classe P

Problèmes pour lesquels il existe un algorithme **Polynomial**

- Complexité de X $\leq \mathcal{O}(n^k)$ avec
  - $n =$ taille des données de X en entrée
  - $k =$ constante indépendante de n

Exemples de problèmes de la classe P

- Rechercher un élément dans un tableau
- Trier un tableau, un fichier, une liste. . .
- Rechercher un plus court chemin entre 2 sommets d'un graphe
- . . .

Classes de complexité

La classe P

Problèmes pour lesquels il existe un algorithme **Polynomial**

- Complexité de X $\leq \mathcal{O}(n^k)$ avec
  - $n =$ taille des données de X en entrée
  - $k =$ constante indépendante de n

Exemples de problèmes de la classe P

- Rechercher un élément dans un tableau
- Trier un tableau, un fichier, une liste...
- Rechercher un plus court chemin entre 2 sommets d'un graphe
- ...

Classes de complexité

La classe NP

- Problèmes pour lesquels il existe un algorithme Polynomial pour une machine de Turing **Non** déterministe !
- Machine de Turing non déterministe
  - exécute un nombre fini d'alternatives en parallèle
  - $X \in NP \Rightarrow$ vérif$(X) \in P$
    où vérif(X) = décider si une donnée est solution de X

Relation entre P et NP

- $P \subseteq NP$
- Conjecture : $P \neq NP$
  ... 1 million de dollars à gagner !
  (claymath.org/millennium-problems/p-vs-np-problem)

Classes de complexité

La classe NP

- Problèmes pour lesquels il existe un algorithme Polynomial pour une machine de Turing **Non** déterministe !
- Machine de Turing non déterministe
  - exécute un nombre fini d'alternatives en parallèle
  - $X \in NP \Rightarrow$ vérif(X) $\in P$
    où vérif(X) = décider si une donnée est solution de X

Relation entre P et NP

- $P \subseteq NP$
- Conjecture : $P \neq NP$
  ... 1 million de dollars à gagner !
  (`claymath.org/millennium-problems/p-vs-np-problem`)

Classes de complexité

Problèmes NP-complets

- Les problèmes les plus difficiles de la classe NP
- X est NP-complet s'il appartient à NP et qu'il existe une réduction polynomiale de tout problème de NP à X
- Conséquence : X est NP-complet $\rightsquigarrow$ X $\in$ P $\Rightarrow$ P=NP
- Théorème de [Cook 1971] : SAT est NP-complet

Démonstration de NP-complétude

- Montrer que le problème X appartient à NP
- Trouver une procédure polynomiale pour transformer un problème NP-complet en X

Classes de complexité

Problèmes NP-complets

- Les problèmes les plus difficiles de la classe NP
- X est NP-complet s'il appartient à NP et qu'il existe une réduction polynomiale de tout problème de NP à X
- Conséquence : X est NP-complet $\rightsquigarrow$ X $\in$ P $\Rightarrow$ P=NP
- Théorème de [Cook 1971] : SAT est NP-complet

Démonstration de NP-complétude

- Montrer que le problème X appartient à NP
- Trouver une procédure polynomiale pour transformer un problème NP-complet en X

Classes de complexité

Problèmes NP-difficiles

Les problèmes au moins aussi difficiles que ceux de NP

- X est NP-difficile s'il existe une réduction polynomiale de tout problème de NP à X
  (mais X n'est pas forcément dans NP)
- Conséquence : X est NP-difficile $\rightsquigarrow$ X $\in$ P $\Rightarrow$ P=NP
- NP-complet $\subset$ NP-difficile

Problèmes indécidables

Problèmes pour lesquels il n'existe pas d'algorithme

- Exemples : problème de l'arrêt

Pour en savoir plus

Le zoo des complexités : `complexityzoo.com`

Classes de complexité

Problèmes NP-difficiles

Les problèmes au moins aussi difficiles que ceux de NP

- X est NP-difficile s'il existe une réduction polynomiale de tout problème de NP à X
  (mais X n'est pas forcément dans NP)
- Conséquence : X est NP-difficile $\rightsquigarrow$ X $\in$ P $\Rightarrow$ P=NP
- NP-complet $\subset$ NP-difficile

Problèmes indécidables

Problèmes pour lesquels il n'existe pas d'algorithme

- Exemples : problème de l'arrêt

Pour en savoir plus

Le zoo des complexités : `complexityzoo.com`

Classes de complexité

Problèmes NP-difficiles

Les problèmes au moins aussi difficiles que ceux de NP

- X est NP-difficile s'il existe une réduction polynomiale de tout problème de NP à X
  (mais X n'est pas forcément dans NP)
- Conséquence : X est NP-difficile $\rightsquigarrow$ X $\in$ P $\Rightarrow$ P=NP
- NP-complet $\subset$ NP-difficile

Problèmes indécidables

Problèmes pour lesquels il n'existe pas d'algorithme

- Exemples : problème de l'arrêt

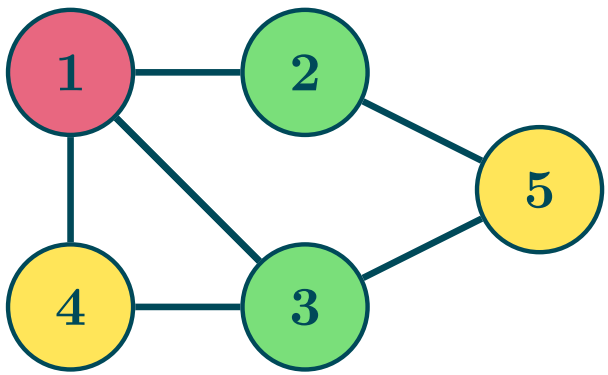
Pour en savoir plus

Le zoo des complexités : `complexityzoo.com`

Complexité des problèmes d'optimisation

Complexité optimisation

- Classes de complexité définies pour les problèmes de décision
  - Réponse = oui ou non
  - Exemple : Peut-on colorier ce graphe avec 3 couleurs ?
- Problèmes d'optimisation
  - Réponse = valeur qui optimise une fonction objectif donnée
  - Exemple : Plus petit nombre de couleurs permettant de colorier ce graphe ?
- Problème de décision associé à un problème d'optimisation
  - Existe-t-il une meilleure solution ?
  - Exemple : Peut-on colorier ce graphe avec moins de 3 couleurs ?
- Complexité du problème d'optimisation / problème de décision
  - si le problème de décision est NP-complet, alors le problème d'optimisation est dit NP-difficile



Difficulté des problèmes NP-complets/difficiles

Difficulté des problèmes

- Croissance exponentielle ? !

n	2^n	Temps (si 10^9 instr/s)
30	$\approx 10^9$	≈ 1 seconde
40	$\approx 10^{12}$	≈ 16 minutes
50	$\approx 10^{15}$	≈ 11 jours
60	$\approx 10^{18}$	≈ 32 ans
70	$\approx 10^{21}$	≈ 317 siècles

- En théorie, on ne pourra traiter des données de taille > 50
- En pratique, on peut souvent faire bien mieux. . .
- De très nombreux problèmes réels sont NP-difficiles
  - Mesurer la similarité d'objets
  - Extraire des connaissances à partir de données
  - Trouver un plan d'action pour atteindre un objectif
  - Construire un emploi du temps
  - . . .

Sommaire

Exemple 1 : le problème SAT

Instance du problème SAT

- Formule booléenne sous forme clausale
- Exemple :
  $(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$

Problème de décision

- Peut-on satisfaire toutes les clauses ?
- NP-complet dans le cas général . . . mais polynomial si toutes les clauses ont au plus 2 littéraux

Problème d'optimisation

- Plus grand nombre de clauses pouvant être satisfaites ?
- NP-difficile dans le cas général

Espace de recherche

Ensemble des valuations possibles $\rightsquigarrow \mathcal{O}(2^n)$ si n variables

Exemple 1 : le problème SAT

Instance du problème SAT

- Formule booléenne sous forme clausale
- Exemple :
  $(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$

Problème de décision

- Peut-on satisfaire toutes les clauses ?
- NP-complet dans le cas général . . . mais polynomial si toutes les clauses ont au plus 2 littéraux

Problème d'optimisation

- Plus grand nombre de clauses pouvant être satisfaites ?
- NP-difficile dans le cas général

Espace de recherche

Ensemble des valuations possibles $\rightsquigarrow \mathcal{O}(2^n)$ si n variables

Exemple 1 : le problème SAT

Instance du problème SAT

- Formule booléenne sous forme clausale
- Exemple :
  $(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$

Problème de décision

- Peut-on satisfaire toutes les clauses ?
- NP-complet dans le cas général . . . mais polynomial si toutes les clauses ont au plus 2 littéraux

Problème d'optimisation

- Plus grand nombre de clauses pouvant être satisfaites ?
- NP-difficile dans le cas général

Espace de recherche

Ensemble des valuations possibles $\rightsquigarrow \mathcal{O}(2^n)$ si n variables

Exemple 1 : le problème SAT

Instance du problème SAT

- Formule booléenne sous forme clausale
- Exemple :
  $(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$

Problème de décision

- Peut-on satisfaire toutes les clauses ?
- NP-complet dans le cas général . . . mais polynomial si toutes les clauses ont au plus 2 littéraux

Problème d'optimisation

- Plus grand nombre de clauses pouvant être satisfaites ?
- NP-difficile dans le cas général

Espace de recherche

Ensemble des valuations possibles $\rightsquigarrow \mathcal{O}(2^n)$ si n variables

Exemple 1 : le problème SAT

Exemple

- $(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$
- $2^5 = 32$ valuations

a	b	c	d	e
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
...				
1	1	1	1	1

Exemple 2 : les CSPs

Définition

Problème de Satisfaction de Contraintes (CSP) défini par (X, D, C)

- $X = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble des variables
- $D(X_i)$ est le domaine de X_i
- C est l'ensemble des contraintes du problème

Exemple

- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_i) = \{0, 1\}$, pour toute variable $X_i \in X$
- $C = \{X_1 \neq X_2, X_3 \neq X_4, X_1 + X_3 < X_2\}$

Exemple 2 : les CSPs

Définition

Problème de Satisfaction de Contraintes (CSP) défini par (X, D, C)

- $X = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble des variables
- $D(X_i)$ est le domaine de X_i
- C est l'ensemble des contraintes du problème

Exemple

- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_i) = \{0, 1\}$, pour toute variable $X_i \in X$
- $C = \{X_1 \neq X_2, X_3 \neq X_4, X_1 + X_3 < X_2\}$

Exemple 2 : les CSPs

Problème de décision

- Peut-on satisfaire toutes les contraintes ?
  $\rightsquigarrow$ trouver une affectation satisfaisant les contraintes
- NP-complet dans le cas général . . . polynomial dans certains cas (ex : contraintes linéaires / réels) . . . parfois indécidable (ex : contraintes non linéaires / réels)

Problèmes d'optimisation

- Plus grand nombre de contraintes pouvant être satisfaites ?
- CSP valués
  $\rightsquigarrow$ Expression de préférences entre contraintes
  $\rightsquigarrow$ Chercher à optimiser ces préférences
- NP-difficile dans le cas général

Espace de recherche (cas général)

Ensemble des affectations possibles

- k^n valuations si $|X| = n$ et $|D(X_i)| = k$, $\forall X_i \in X$

Exemple 2 : les CSPs

Problème de décision

- Peut-on satisfaire toutes les contraintes ?
  $\rightsquigarrow$ trouver une affectation satisfaisant les contraintes
- NP-complet dans le cas général . . . polynomial dans certains cas (ex : contraintes linéaires / réels) . . . parfois indécidable (ex : contraintes non linéaires / réels)

Problèmes d'optimisation

- Plus grand nombre de contraintes pouvant être satisfaites ?
- CSP valués
  $\rightsquigarrow$ Expression de préférences entre contraintes
  $\rightsquigarrow$ Chercher à optimiser ces préférences
- NP-difficile dans le cas général

Espace de recherche (cas général)

Ensemble des affectations possibles

- k^n valuations si $|X| = n$ et $|D(X_i)| = k$, $\forall X_i \in X$

Exemple 2 : les CSPs

Problème de décision

- Peut-on satisfaire toutes les contraintes ?
  $\rightsquigarrow$ trouver une affectation satisfaisant les contraintes
- NP-complet dans le cas général . . . polynomial dans certains cas (ex : contraintes linéaires / réels) . . . parfois indécidable (ex : contraintes non linéaires / réels)

Problèmes d'optimisation

- Plus grand nombre de contraintes pouvant être satisfaites ?
- CSP valués
  $\rightsquigarrow$ Expression de préférences entre contraintes
  $\rightsquigarrow$ Chercher à optimiser ces préférences
- NP-difficile dans le cas général

Espace de recherche (cas général)

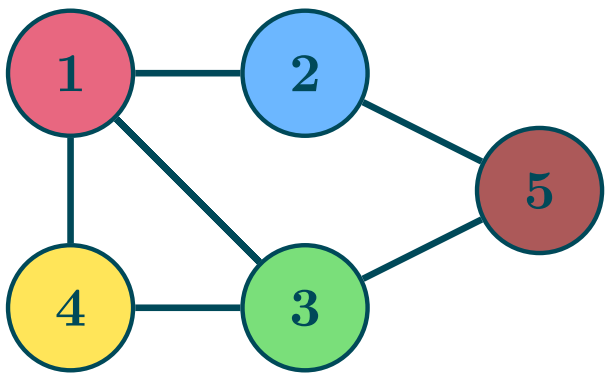
Ensemble des affectations possibles

- k^n valuations si $|X| = n$ et $|D(X_i)| = k$, $\forall X_i \in X$

Exemple 3 : Problèmes dans les graphes

Exemples de problèmes polynomiaux sur les graphes

- Déterminer les sous-ensembles de sommets connectés
- Trouver le plus court chemin entre 2 sommets
- Trouver un chemin passant 1 fois par chaque arc (Eulérien)
- Connecter un ensemble de sommets au plus faible coût
- Maximiser un flot de véhicules dans un réseau de transport



Exemple 3 : Problèmes dans les graphes

Exemples de problèmes NP-complets/difficiles

- Trouver un plus court cycle hamiltonien
  $\rightsquigarrow$ Tournée d'un voyageur de commerce
- Trouver k sommets connectés 2 à 2 par des arêtes
  $\rightsquigarrow$ Problème de la clique
- Colorier les sommets
  $\rightsquigarrow$ Allouer des ressources/contraintes d'exclusion
- Partitionner les sommets
  $\rightsquigarrow$ Classification
- Comparer des graphes
  $\rightsquigarrow$ Problèmes d'appariements de graphes :
  Isomorphisme (équivalence, isomorphique-complet), Isomorphisme de sous-graphe (inclusion, NP-complet), Plus grand sous-graphe commun (intersection, NP-difficile), Meilleur appariement multivoque (similarité/opérations de déformation, NP-difficile)

Exemple 4 : Problèmes de planification

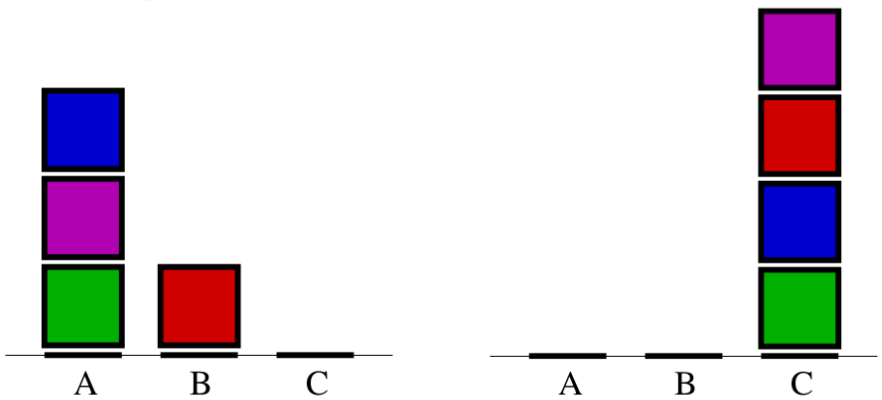
Instance d'un problème de planification

- Définie par
  - Un ensemble (éventuellement infini) d'états E
  - Un état initial $E_0 \in E$
  - Un ensemble d'états finaux $F \subset E$
  - Un ensemble d'opérations O permettant de passer d'états en états $E_i \xrightarrow{O_k} E_j$ si O_k permet de passer de E_i à E_j
- Un plan est une suite d'opérations permettant de passer de l'état initial E_0 à un état final de F

  Plan $= E_0 \xrightarrow{O_1} E_1 \xrightarrow{O_2} E_2 \dots E_{n-1} \xrightarrow{O_n} E_n$ avec $E_n \in F$

Exemple 4 : Problèmes de planification

Exemple : le monde des blocs



Etat initial Etat final

Opérations = prendre un cube au sommet d'une pile et le poser au sommet d'une autre pile

Exemple 4 : Problèmes de planification

Problème de décision

- Existe-t-il un plan ?
- Algorithme polynomial / graphe G = (E, O)
  . . . mais le nombre d'états est généralement exponentiel !

Problèmes d'optimisation

- Trouver le plus petit plan / nombre d'opérations
- Trouver le meilleur plan / objectif donné

Exemple 4 : Problèmes de planification

Problème de décision

- Existe-t-il un plan ?
- Algorithme polynomial / graphe G = (E, O)
  . . . mais le nombre d'états est généralement exponentiel !

Problèmes d'optimisation

- Trouver le plus petit plan / nombre d'opérations
- Trouver le meilleur plan / objectif donné

Exemple 5 : Recherche d'ensembles fréquents

Ensembles fréquents

- Contexte : Extraction de Connaissances à partir de Données
- Exemple de données :
  - $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble d'attributs (items)
  - $D = \{T_1, T_2, \ldots, T_k\}$ un ensemble de transactions tq $T_j \subset I$

attributs :	i_1	i_2	i_3	i_4
T_1	$\times$		$\times$	
T_2	$\times$	$\times$	$\times$	
T_3		$\times$		$\times$
T_4		$\times$	$\times$	$\times$
T_5	$\times$	$\times$	$\times$	

- Exemples de "connaissances" :
  - Ensembles fréquents : $\{i_1, i_3\}$ a une fréquence $\geq 3/5$
  - Règles d'association : $\{i_1, i_3\} \Rightarrow \{i_2\}[3/5, 2/3]$
- Espace de recherche = ensemble des sous-ensembles de I
  $\rightsquigarrow 2^n$ sous-ensembles possibles si $|I| = n$
  $\rightsquigarrow \ldots$ le nombre k de transactions dans D est souvent très grand

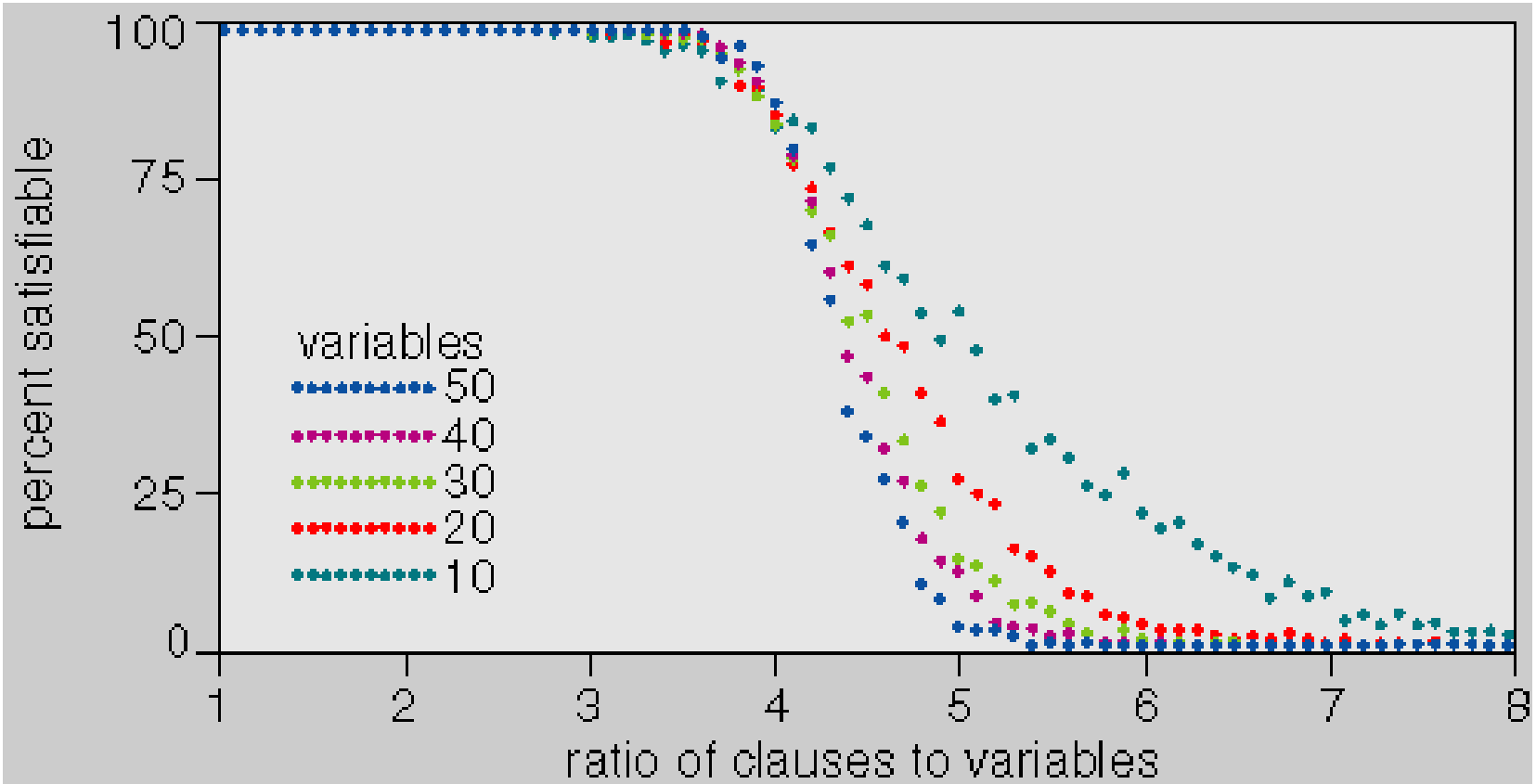
Sommaire

Transition de phases

Transition de phases

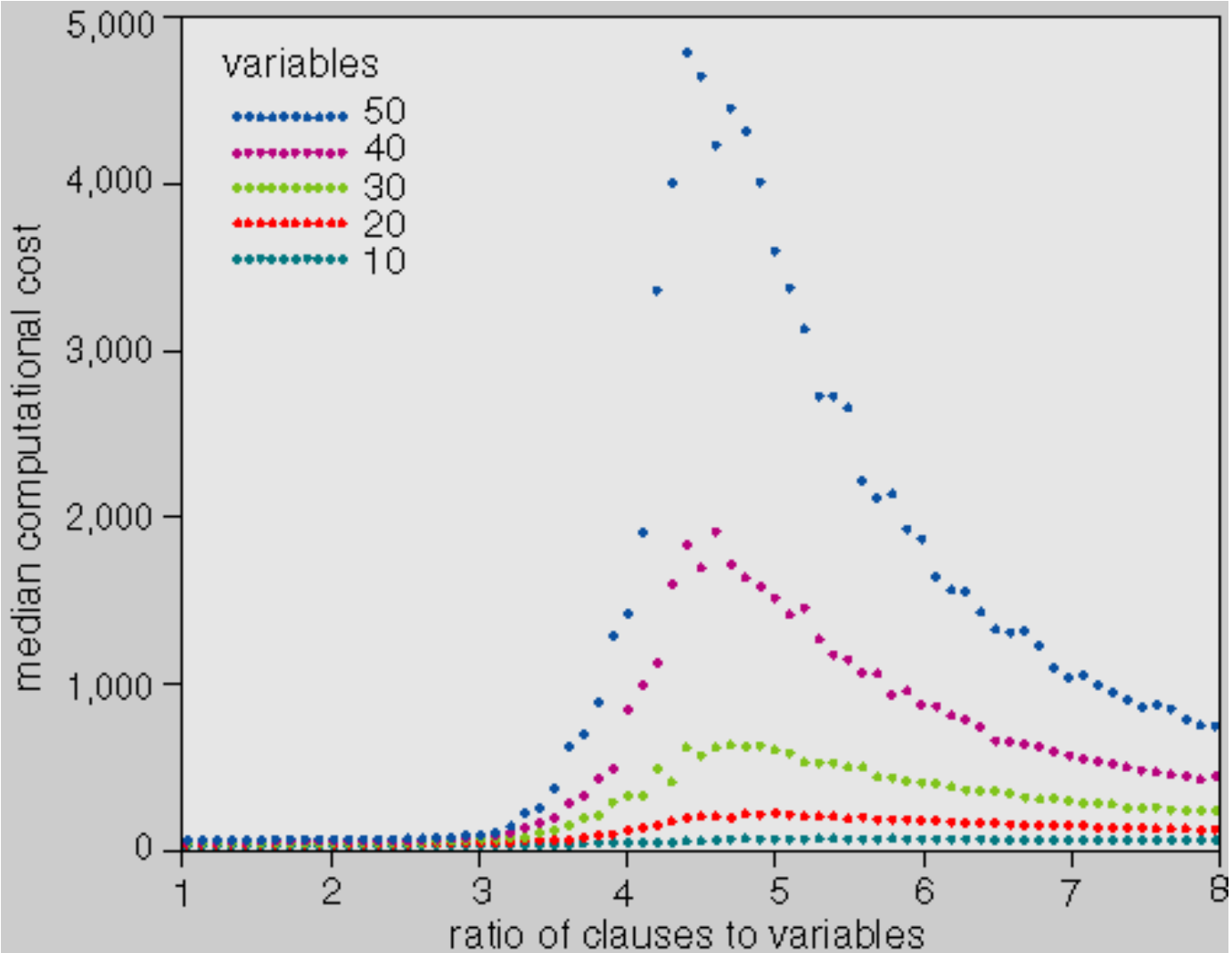
- Toutes les instances d'un problème ne sont pas de difficultés égales
- Exemple : Décider de la satisfiabilité d'une instance SAT donnée
  - La difficulté dépend du nombre n de variables...
    $\rightsquigarrow$ taille de l'espace de recherche $= 2^n$
  - ... mais aussi du nombre p de clauses
    - Peu de clauses $\Rightarrow$ Instance sous-contrainte
      $\rightsquigarrow$ Solution trouvée facilement
    - Beaucoup de clauses $\Rightarrow$ Instance sur-contrainte
      $\rightsquigarrow$ Inconsistance facilement montrée... ou solution facilement trouvée
    - entre les 2 $\Rightarrow$ ça se complique ! ! !
- Etude sur des instances de 3-SAT (3 littéraux par clause)
  - Génération aléatoire d'instances, en fonction
    - du nombre n de variables ($n \in \{10, 20, 30, 40, 50\}$),
    - du nombre p de clauses ($n \leq p \leq 8n$)
  - Étude du taux d'instances satisfiables en fonction du ratio p/n
  - Étude du temps de résolution en fonction du ratio p/n

Transition de phases



(image empruntée à des transparents de Toby Walsh)

Transition de phases



(image empruntée à des transparents de Toby Walsh)

Transition de phases

Transition de phases

- Pour 3-SAT : changement abrupte entre satisfiable et non satisfiable quand $\frac{nbclauses}{nbvariables} = 4.3$
- Ce phénomène est appelé transition de phases
  - Correspond à un pic de difficulté pour la résolution
  - Apparaît aussi pour de très nombreux autres problèmes
  - Indépendant de l'algorithme utilisé

Transition de phases

Où se trouve la transition de phases ?

- Là où la probabilité d'être satisfiable est de 50%
- Estimer le taux de constrainedness κ d'une instance :
  - soit 2^n la taille de l'espace de recherche de l'instance
  - soit p le nombre estimé de solutions de l'instance
  - $\kappa = 1 - \frac{log_2(p)}{n}$
- La transition de phases se trouve là où $\kappa = 1$:
  - Si $\kappa \approx 0$ alors l'instance est sous-contrainte
  - Si $\kappa \approx 1$ alors l'instance est critique
  - Si $\kappa \approx \infty$ alors l'instance est sur-contrainte

Transition de phases

A quoi ça sert ?

- A tester son nouvel algorithme sur les instances vraiment difficiles
- A concevoir de meilleurs algorithmes
- ...

Pour en savoir plus

- Where the really hard problems are ?
  P. Cheeseman, B. Kanefsky et W.M. Taylor, [IJCAI'1991]

Sommaire

Sommaire

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacité.

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacité.

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacité.

Résolution de problèmes NP-complets/difficiles

Exploration exhaustive de l'espace de recherche

- Structurer l'espace de recherche en arbre ou en treillis
- Filtrer et/ou borner $\rightsquigarrow$ couper des zones
- Heuristiques $\rightsquigarrow$ explorer en premier les zones prometteuses

$\rightsquigarrow$ Approches correctes et complètes... Complexité exponentielle

Exploration opportuniste de l'espace de recherche

- Approches basées sur le voisinage
  Recherche locale, Algorithmes génétiques
- Approches constructives
  Algorithmes gloutons, fourmis, EDA

$\rightsquigarrow$ Approches incomplètes... Complexité polynomiale

Résolution de problèmes NP-complets/difficiles

Exploration exhaustive de l'espace de recherche

- Structurer l'espace de recherche en arbre ou en treillis
- Filtrer et/ou borner $\rightsquigarrow$ couper des zones
- Heuristiques $\rightsquigarrow$ explorer en premier les zones prometteuses

$\rightsquigarrow$ Approches correctes et complètes... Complexité exponentielle

Exploration opportuniste de l'espace de recherche

- Approches basées sur le voisinage
  Recherche locale, Algorithmes génétiques
- Approches constructives
  Algorithmes gloutons, fourmis, EDA

$\rightsquigarrow$ Approches incomplètes... Complexité polynomiale

Résolution de problèmes NP-complets/difficiles

Résolution approximée

- Calcul d'une approximation de la solution optimale
  $\rightsquigarrow$ erreur bornée
- Temps de calcul dépend de l'erreur : erreur $\searrow$ temps $\nearrow$
  $\rightsquigarrow$ erreur bornée

$\rightsquigarrow$ Approches non correctes... Complexité polynomiale

Pas étudié dans ce cours.

- Pour en savoir plus :
  `http://legacy.orie.cornell.edu/~dpw/cornell.ps`

Principe général des approches complètes

Approches top-down : construction d'un arbre de recherche

$\rightsquigarrow$ découper l'espace de recherche en portions de + en + petites

- racine $\rightsquigarrow$ espace de recherche initial
- noeud $\rightsquigarrow$ portion de l'espace de recherche
- 2 types de feuilles :
  - feuille "succès" $\rightsquigarrow$ solution du problème
  - feuille "échec" $\rightsquigarrow$ portion sans solution

Approches bottom-up : construction d'un treillis

$\rightsquigarrow$ construire des "solutions" de plus en plus grandes

- niveau 1 du treillis $\rightsquigarrow$ solutions "de taille 1"
- niveau k du treillis $\rightsquigarrow$ solutions "de taille k"
- haut du treillis $\rightsquigarrow$ solutions "de plus grande taille"

Pas étudié dans ce cours.

Principe général des approches complètes

Approches top-down : construction d'un arbre de recherche

$\rightsquigarrow$ découper l'espace de recherche en portions de + en + petites

- racine $\rightsquigarrow$ espace de recherche initial
- noeud $\rightsquigarrow$ portion de l'espace de recherche
- 2 types de feuilles :
  - feuille "succès" $\rightsquigarrow$ solution du problème
  - feuille "échec" $\rightsquigarrow$ portion sans solution

Approches bottom-up : construction d'un treillis

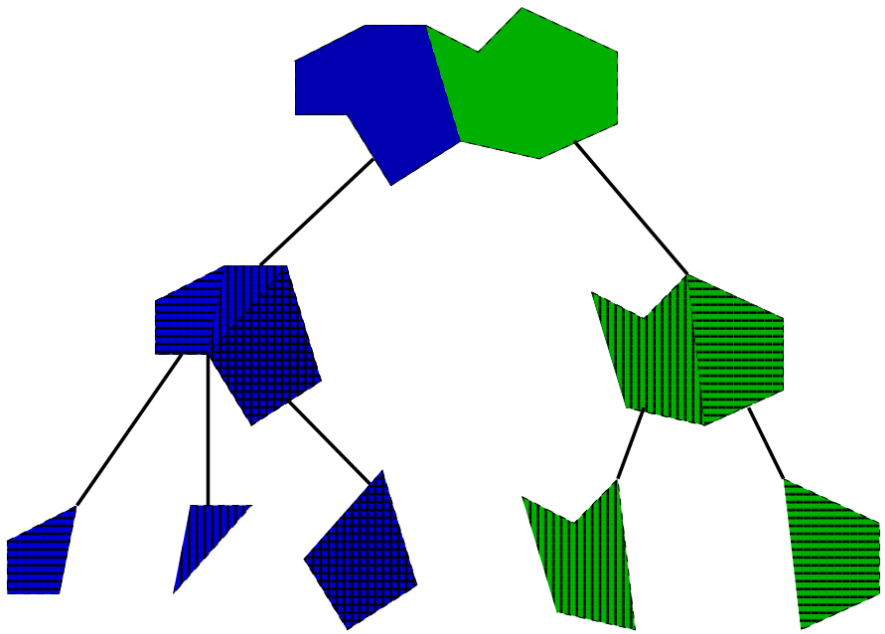
$\rightsquigarrow$ construire des "solutions" de plus en plus grandes

- niveau 1 du treillis $\rightsquigarrow$ solutions "de taille 1"
- niveau k du treillis $\rightsquigarrow$ solutions "de taille k"
- haut du treillis $\rightsquigarrow$ solutions "de plus grande taille"

Pas étudié dans ce cours.

Sommaire

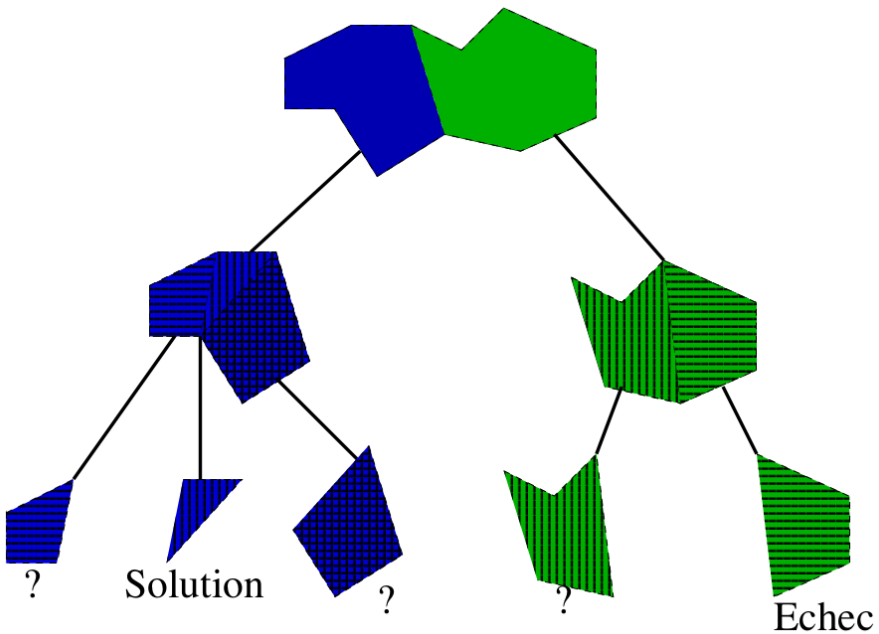
Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces

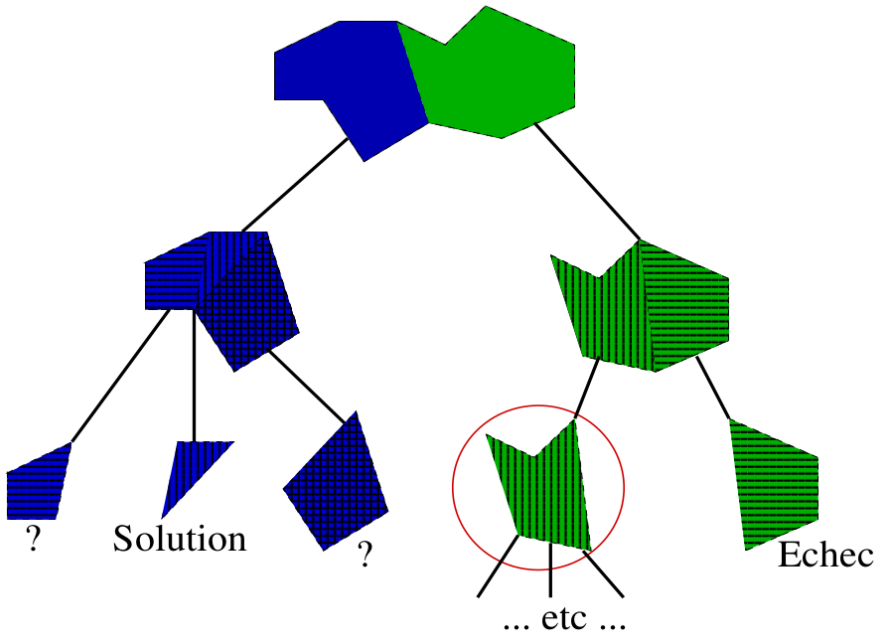
Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces
- évaluation $\rightsquigarrow$ solution, échec ou ?
  complexité (faiblement) polynomiale

Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces
- évaluation $\rightsquigarrow$ solution, échec ou ?
  complexité (faiblement) polynomiale
- sélection $\rightsquigarrow$ prochain noeud à développer

Exemple 1 : Programmation Linéaire en Nombres Entiers

Exemple de PLNE

maximiser $f = 3x + 4y - 2z$
tel que $2x + z \geq 10$
$x - 3y \leq 30$
et x, y et z sont des entiers

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par l'algorithme du simplex ou du point intérieur
- ... mais NP-difficile avec la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par Séparation & évaluation

Exemple 1 : Programmation Linéaire en Nombres Entiers

Exemple de PLNE

maximiser $f = 3x + 4y - 2z$
tel que $2x + z \geq 10$
$x - 3y \leq 30$
et x, y et z sont des entiers

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par l'algorithme du simplex ou du point intérieur
- ... mais NP-difficile avec la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par Séparation & évaluation

Exemple 1 : Programmation Linéaire en Nombres Entiers

Résolution d'un PLNE par Séparation & Évaluation

- Fonction d'évaluation :
  Relâcher la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$
  - si x^* ,y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche
- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2 (choisir une variable ayant une valeur non entière dans f^*)

Pour en savoir plus

`http://mat.gsia.cmu.edu/orclass/integer/integer.html`

Exemple 1 : Programmation Linéaire en Nombres Entiers

Résolution d'un PLNE par Séparation & Évaluation

- Fonction d'évaluation :
  Relâcher la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$
  - si x^* ,y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche
- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2 (choisir une variable ayant une valeur non entière dans f^*)

Pour en savoir plus

`http://mat.gsia.cmu.edu/orclass/integer/integer.html`

Exemple 1 : Programmation Linéaire en Nombres Entiers

Résolution d'un PLNE par Séparation & Évaluation

- Fonction d'évaluation :
  Relâcher la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$
  - si x^* ,y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche
- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2 (choisir une variable ayant une valeur non entière dans f^*)

Pour en savoir plus

`http://mat.gsia.cmu.edu/orclass/integer/integer.html`

Exemple 1 : Programmation Linéaire en Nombres Entiers

Résolution d'un PLNE par Séparation & Évaluation

- Fonction d'évaluation :
  Relâcher la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$
  - si x^* ,y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche
- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2 (choisir une variable ayant une valeur non entière dans f^*)

Pour en savoir plus

`http://mat.gsia.cmu.edu/orclass/integer/integer.html`

Exemple 2 : Problèmes de planification

Problèmes de planification (rappel)

- Définis par
  - Un ensemble (éventuellement infini) d'états E
  - Un état initial $E_0 \in E$
  - Un ensemble d'états finaux $F \subseteq E$
  - Un ensemble d'opérations O permettant de passer d'états en états $E_i \xrightarrow{O_k} E_j$ si O_k permet de passer de E_i à E_j
- Un plan est une suite d'opérations permettant de passer de l'état initial E_0 à un état final de F

  Plan $= E_0 \xrightarrow{O_1} E_1 \xrightarrow{O_2} E_2 \ldots E_{n-1} \xrightarrow{O_n} E_n$ avec $E_n \in F$
- Problème = trouver le plan de coût minimal
- Espace de recherche = ensemble des suites d'opérations partant de E_0

Exemple 2 : Problèmes de planification

Recherche du plan de coût minimal (algorithme A^*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n) =$ coût des opérations faites de la racine jusqu'à n
  - $h(n) =$ borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$
  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n
  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ nœud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification

Recherche du plan de coût minimal (algorithme A^*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n) =$ coût des opérations faites de la racine jusqu'à n
  - $h(n) =$ borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$
  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n
  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ nœud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification

Recherche du plan de coût minimal (algorithme A^*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n) =$ coût des opérations faites de la racine jusqu'à n
  - $h(n) =$ borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ nœud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification

Recherche du plan de coût minimal (algorithme A^*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n) =$ coût des opérations faites de la racine jusqu'à n
  - $h(n) =$ borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ nœud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification

- Exemple de problème de planification : le "taquin"
  - État initial =

	B	C
A	D	F
G	E	H

  - État final =

A	B	C
D	E	F
G	H	

  - Opération = échanger la case vide avec une case adjacente
  - But = trouver le plus petit plan (en nombre d'opérations)
- Coût du début du plan = nombre d'opérations déjà effectuées
- Estimation du coût pour atteindre l'état final = ? ? ?

B	D	C
	A	F
G	E	H

Exemple 2 : Problèmes de planification

- Exemple de problème de planification : le "taquin"
  - État initial =

	B	C
A	D	F
G	E	H

  - État final =

A	B	C
D	E	F
G	H	

  - Opération = échanger la case vide avec une case adjacente
  - But = trouver le plus petit plan (en nombre d'opérations)
- Coût du début du plan = nombre d'opérations déjà effectuées
- Estimation du coût pour atteindre l'état final = distance de Manhattan

B	D	C
	A	F
G	E	H

$\delta(A) = 2$, $\delta(B) = 1$, $\delta(C) = 0$, $\delta(D) = 2$, $\delta(E) = 1$, $\delta(F) = 0$, $\delta(G) = 0$, $\delta(H) = 1 \Rightarrow$ au moins 7 opérations pour aller jusqu'à l'état final

Sommaire

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming :

the user states the problem,
the computer solves it."

Eugene C. Freuder

La programmation par Contraintes (PPC)

- Spécifier le problème en termes de contraintes
  $\rightsquigarrow$ Problèmes de satisfaction de contraintes (CSPs)
- Concevoir des algorithmes de résolution de CSPs $\rightsquigarrow$ Solveurs de contraintes
- Intégrer ces solveurs dans un langage
  $\rightsquigarrow$ Le programmeur définit les contraintes...
  ... et le solveur cherche la solution.
  Exemples de bibliothèques de PPC :
  - ALICE [Jean-Louis Laurière, 1976]
  - CHIP, Prolog V, Gnu-Prolog
  - CHOCO, IBM Ilog solver, Gecode, LocalSolver, Glucose, Toulbar
  - ...
- Pas toujours aussi efficace qu'un programme "cousu main"
  ... mais tellement plus vite développé !

Programme des reines avec Gecode

```
using namespace Gecode;
class Nqueens : public Space {
protected :
        IntVarArray l;
public :
        Queens(int n) : l(*this, n, 0, n − 1) {
                //post constraints
                distinct(*this,l);
                distinct(*this, IntArgs : :create(n, 0, 1), l);
                distinct(*this, IntArgs : :create(n, 0, −1), l);
                //post branching
                branch(*this, l, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
                }
        //search support
        Queens(bool share, Queens & s) : Space(share, s) {
                l.update(*this, share, s.l);
        }
        virtual Space* copy(bool share) {
                return new Queens(share,*this);
        }
        //print solution
        void print(void) const {
                std : :cout << l << std : :endl;
        }
};
```

Programme des reines avec Gecode

```
using namespace Gecode;
int main(int argc, char* argv[]) {
// create model and search engine
Queens * m = new Queens(5);
DFS<Queens> e(m);
Queens * s = e.next();
while(s) {
        s->print();
        delete s;
        s = e.next();
}
std::cout << "no more solution" << std::endl;
return 0;
}
```

Problèmes de Satisfaction de Contraintes

Définition d'un CSP (rappel)

CSP $= (X, D, C)$ tel que

- $X = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble des variables
- $D(X_i)$ est le domaine de X_i
- C est l'ensemble des contraintes du problème

Solution d'un CSP (rappel)

- Affectation = ensemble de couples $< X_i, v_i >$ tel que $v_i \in D(X_i)$
- Affectation partielle $\rightsquigarrow$ certaines variables ne sont pas affectées
- Affectation totale $\rightsquigarrow$ toutes les variables sont affectées
- Affectation consistante $\rightsquigarrow$ contraintes satisfaites

Solution = affectation complète consistante

Problèmes de Satisfaction de Contraintes

"Résoudre" un CSP (X, D, C) peut signifier :

- trouver une solution,
- prouver qu'il existe au moins une solution (consistance),
- trouver les intervalles de valeurs dans lesquels se trouvent les solutions,
- trouver toutes les solutions (si énumérable),
- trouver la "meilleure" solution. . .
  . . . par rapport à une fonction objectif donnée
- trouver une affectation qui maximise le nombre de contraintes satisfaites (MAX-CSPs, CSPs valués, Contraintes "soft", . . .)

Techniques de résolution de CSPs

Double origine

Intelligence Artificielle et Recherche Opérationnelle

Dépend du type des domaines et contraintes

- Domaines continus (réels) / contraintes numériques
  Algorithme de Gauss, Simplex, bases de Grobner...
  Arithmétique des intervalles
- Domaines discrets (énumérables $\rightarrow$ entiers)
  $\rightsquigarrow$ résolution par Séparation & Propagation
- Domaines symboliques (chaînes, graphes,...)
  $\rightsquigarrow$ algorithmes "dédiés"

Résolution par Séparation & Propagation

Construction d'un arbre de recherche

- Racine = affectation vide
- Nœuds internes = affectations partielles consistantes
  - Affectations partielles inconsistantes $\rightsquigarrow$ échecs
  - Affectations totales consistantes $\rightsquigarrow$ solutions

Fonction de séparation

$\rightsquigarrow$ Séparer le domaine courant d'une variable en plusieurs parties

- choisir une variable dont le domaine contient plusieurs valeurs
- découper le domaine en sous-domaines et créer autant de fils que de sous-domaines

$\rightsquigarrow$ Heuristique de choix de variable

Résolution par Séparation & Propagation

Fonction de séparation

$\rightsquigarrow$ Séparer le domaine courant d'une variable en 2 parties

- la variable est contrainte à une valeur de son domaine : $X_i = v$
- la variable est contrainte à une valeur différente de la valeur précédente : $X_i \neq v$

$\rightsquigarrow$ Ajout de contraintes $\Rightarrow$ Plus de filtrage

$\rightsquigarrow$ Calcul de nogoods simplifié

Résolution par Séparation & Propagation

Fonction de sélection

$\rightsquigarrow$ Choisir le prochain noeud à développer

- En général : en profondeur d'abord (DFS ou BAB)
  $\rightsquigarrow$ moins coûteux en mémoire
- Heuristique sur l'ordre de développement des fils $\rightsquigarrow$ choisir en premier les valeurs les plus "prometteuses"
- Techniques de retour en arrière intelligent (BJ [Gas79], GBJ [Dec90], CBJ [Pro93]) $\rightsquigarrow$ évitent de nombreuses redondances, mais il y a surcoût

$\rightsquigarrow$ Heuristiques classiques : minDomaine, minDomaine/Degrédynamique, minDomaine/DegréDynamiquePondéré

$\rightsquigarrow$ Mauvais choix de sous-problème $\rightarrow$ Restarts

Résolution par Séparation & Propagation

Fonction de sélection

$\rightsquigarrow$ Mauvais choix de sous-problème $\rightarrow$ Restarts

- Conséquence dramatique : temps exponentiel pour prouver l'absence de solution
- Arrêter l'exploration du sous-problème courant et recommencer depuis la racine
- Utiliser les informations collectées lors de l'exploration du sous-problème
- Combiner avec un choix stochastique

Résolution par Séparation & Propagation

Fonction d'évaluation

$\rightsquigarrow$ Propagation des contraintes

- Vérifier que les variables affectées satisfont toutes les contraintes
  $\rightsquigarrow$ "retour-arrière simple" (Simple Backtrack)
- Filtrer les domaines des variables non affectées
  $\rightsquigarrow$ vérification d'une cohérence locale

Résolution par Séparation & Propagation

Algorithme

$A \leftarrow \emptyset$
tant que *il reste des variables non affectées dans* A **faire**
    choisir une variable X_i non affectée dans A
    choisir une valeur $v \in D(X_i)$
    si *propagation*$(X, C \cup X_i = v, D)$=*échec* **alors**
        "backtrack"

Propagation de contraintes et Cohérences locales

$\rightsquigarrow$ Différents niveaux de cohérence locale

- Cohérence de nœud (complexité : $O(nd)$) :
  $\forall X_i, \forall v \in D(X_i), \{< X_i, v >\} \cup A$ est consistant
- Cohérence d'arc (complexité : $O(md^2)$) :
  $\forall X_i, \forall v \in D(X_i), \forall X_j, \exists u \in D(X_j)$:
  $\{< X_i, v >, < X_j, u >\} \cup A$ est consistant
- k-cohérence (complexité : $O(n^k d^k)$)

Résolution par Séparation & Propagation

Propagation de contraintes et Cohérences locales

$\rightsquigarrow$ Différents niveaux de cohérence locale

- Cohérence de nœud (complexité : $O(nd)$) :
  $\forall X_i, \forall v \in D(X_i), \{< X_i, v >\} \cup A$ est consistant
- Cohérence d'arc (complexité : $O(md^2)$) :
  $\forall X_i, \forall v \in D(X_i), \forall X_j, \exists u \in D(X_j)$:
  $\{< X_i, v >, < X_j, u >\} \cup A$ est consistant
- k-cohérence (complexité : $O(n^k d^k)$)

$\rightsquigarrow$ Extension aux contraintes n-aires de la cohérence d'arc principalement (le bon compromis) : cohérence d'arc généralisée

Résolution par Séparation & Propagation

Propagation de contraintes et Cohérences locales dans Gecode

$\rightsquigarrow$ Propagation à la carte : gestion fine des coûts

- Propagation de domaines : cohérence d'arc
- Propagation de bornes : cohérence d'arc restreinte aux bornes des domaines
- Propagation de valeurs : cohérence de nœud

Résolution par Séparation & Propagation : BAB

Branch And Bound : Optimisation

Un CSP (X, D, C) plus une fonction objectif

- DFS classique
- À chaque nouvelle solution trouvée : ajout d'une contrainte pour trouver une solution meilleure
- Calcul d'une borne inférieure sur la qualité de la solution en cours de construction
  $\rightsquigarrow$ Si la qualité est insuffisante, alors échec

Contraintes souples

Un CSP (X, D, C)

- C : des fonctions de coûts
- BAB + propagation des contraintes souples (calcul d'une borne inférieure)

$\rightsquigarrow$ Bibliothèque de PPC : Toulbar2

Contraintes globales et algorithmes dédiés

Généricité vs Efficacité

Les solveurs de contraintes peuvent résoudre n'importe quel CSP. En contrepartie, ils sont parfois moins efficaces que des approches dédiées.

Notion de contrainte globale

Introduire de nouvelles contraintes pour certains problèmes "typiques" et intégrer au solveur de contraintes des algorithmes dédiés à ces problèmes

Exemples de contraintes globales

- allDiff(L)
- atmost(L, V, N)
- path(G, V_1, V_2)
- graphIsomorphism(G_1, G_2)
- ...

Programme des reines avec Gecode

```
using namespace Gecode;
class Nqueens : public Space {
protected :
        IntVarArray l;
public :
        Queens(int n) : l(*this, n, 0, n − 1) {
                        //post constraints
                        distinct(*this,l);
                        distinct(*this, IntArgs : :create(n, 0, 1), l);
                        distinct(*this, IntArgs : :create(n, 0, −1), l);
                        //post branching
                        branch(*this, l, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
                        }
        //search support
        Queens(bool share, Queens & s) : Space(share, s) {
                        l.update(*this, share, s.l);
        }
        virtual Space* copy(bool share) {
                        return new Queens(share,*this);
        }
        //print solution
        void print(void) const {
                        std : :cout << l << std : :endl;
        }
};
```

Modélisation

Problèmes

- Isomorphisme de deux graphes
- Plus grand sous-graphe commun à deux graphes
- Voyageur de commerce (Plus Petit Chemin Hamiltonien)
- Coloration d'un graphe
- Taille clique maximum d'un graphe
- Sudoku
- Sac à dos

Sommaire

Tentatives d'amélioration

- Approches orientées vers la preuve de cohérence : Decision Repair
- Approches structurelles : garanties théoriques

Decision Repair [JL02,PV04]

- Maintien d'un ensemble de conflits pour chaque variable x
- Inclut toutes les variables instanciées avant x qui sont en conflit pour au moins une valeur de x déjà testée ou qui sont en cause dans l'échec d'une extension d'une affectation contenant x
- En cas d'échec, retour sur une variable (pas nécessairement la dernière) affectée de l'ensemble des conflits de la variable courante

Generic Decision Repair [PV04]

Algorithme 1 : Generic Decision Repair

Entrées : Un CSP P et A une affectation initiale

```
répéter
    cohérent ← Propager(P, A)
    si cohérent alors
        r ← ÉtendreAffectation(P, A)
        si r = 1 alors
            retourner vrai
    sinon
        r ← RéparerAffectation(P, A)
        si r = 0 alors
            retourner faux
jusqu'à Arrêt() ;
retourner ?
```

Generic Decision Repair [PV04]

- Instances de Recherche complète
- Instances de Recherche locale

Generic Decision Repair [PV04]

Définitions de Propager(P, A)

- Test de cohérence arrière : BT
- Maintien d'une forme réduite de cohérence d'arc : FC
- Maintien de la cohérence d'arc : MAC

Espace mémoire requis

- Enregistrer les justifications de la suppression d'une valeur
- Afin de déduire les causes d'un échec (la valeur de certaines variables)
- Taille mémoire nécessaire exponentielle
- Sauvegarder uniquement les justifications en lien avec A

Generic Decision Repair [PV04]

Définitions de ÉtendreAffectation(P, A)

- Heuristique de choix de variables : minimum (domaine/degré)
- ...

Generic Decision Repair [PV04]

Définitions de RéparerAffectation(P, A)

- Retour sur la dernière variable affectée : stratégie classique
- Retour sur la dernière variable affectée en conflit avec la variable courante : BJ
- Retour sur la dernière variable affectée dans le voisinage de la variable courante : GBJ
- Retour sur la dernière variable affectée dans les causes de l'échec : CBJ

Garder l'affectation des variables intermédiaires

- Retour sur la dernière variable affectée dans les causes de l'échec sans désinstancier les variables intermédiaires : DBT [Gins93]
- Retour sur une variable affectée dans les causes de l'échec sans désinstancier les variables intermédiaires : recherche locale
- ...

Generic Decision Repair [PV04]

DR(Maxconflicts)

- Propager : Forward Checking (cohérence d'arc sur un voisinage de la variable courante)
- RéparerAffectation : Retour sur une variable affectée dans les causes de l'échec qui a participé au plus grand nombre de suppressions de valeurs (sans désinstancier les variables intermédiaires)
- Méthode incomplète
- Preuve d'incohérence possible

DR(Minconflicts)

- RéparerAffectation : Retour sur une variable affectée dans les causes de l'échec qui a participé au plus petit nombre de suppressions de valeurs (sans désinstancier les variables intermédiaires)

Generic Decision Repair [PV04]

DR(Maxconflicts)

- Retour rapide sur les mauvais choix
- Trouver une solution au plus vite

DR(Minconflicts)

- Garder le plus grand nombre de causes d'échecs
- Prouver l'incohérence

Évaluation expérimentale sur quelques instances générées aléatoirement.

Approches structurelles

Objectifs

- Exploitation de la structure du problème
- Identification des parties indépendantes
- Résolution séparée des sous-problèmes
- Assurer des garanties théoriques

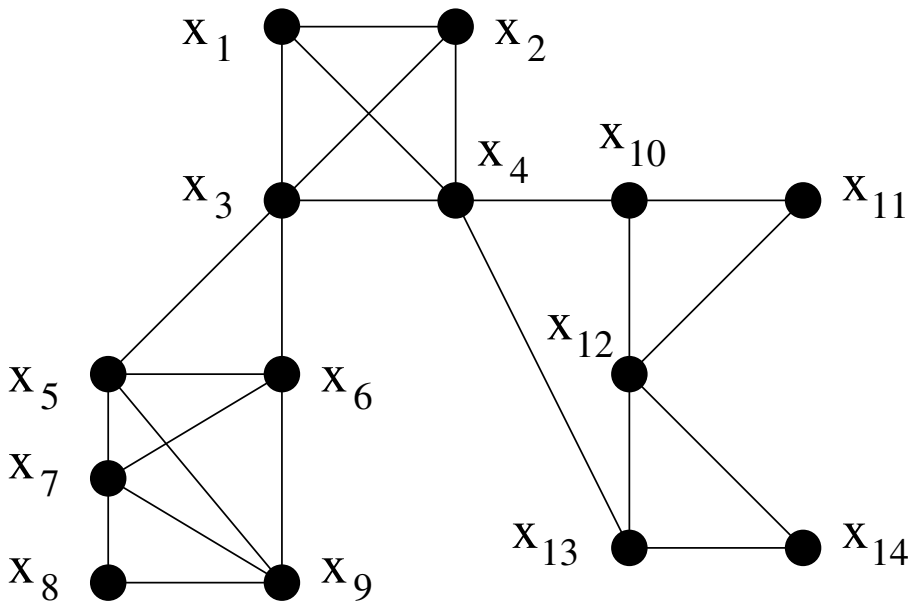
Méthodes

- Pseudo-Tree Search[FQ85]
- Cycle-Cutset [DP87]
- Tree-Clustering[DP89]
- Hypertree-Decomposition[GLS00]
- Recursive Conditioning[Dar01]
- Backtrack on Tree-Decomposition[JT03]
- AndOrSearchGraph[DM07]

Structure d'un CSP

- CSP binaires : graphes de contraintes (X, C)

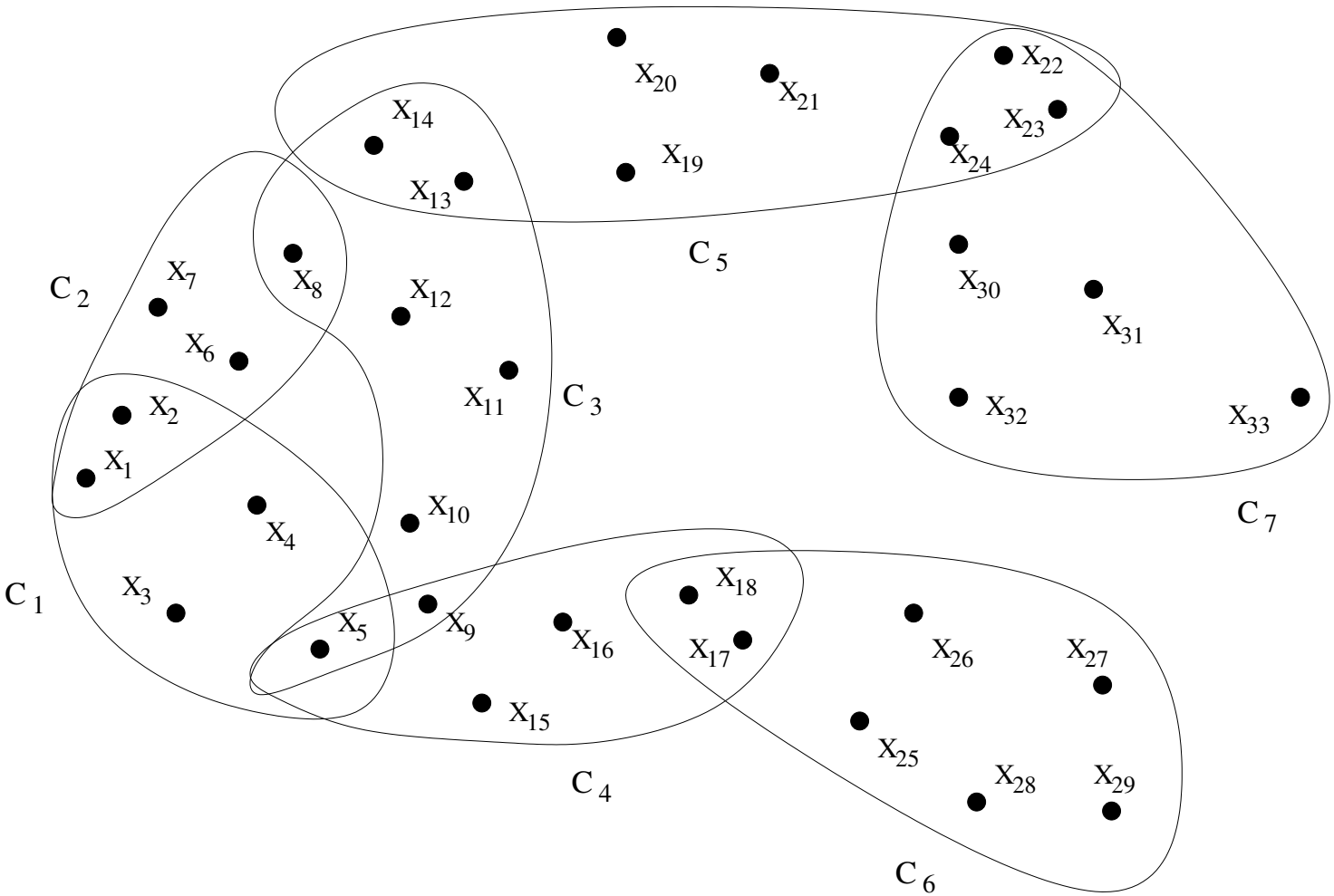
Un graphe de contraintes



Structure d'un CSP

- CSP binaires : graphes de contraintes (X, C)
- CSP n-aires : hypergraphes de contraintes (X, C)

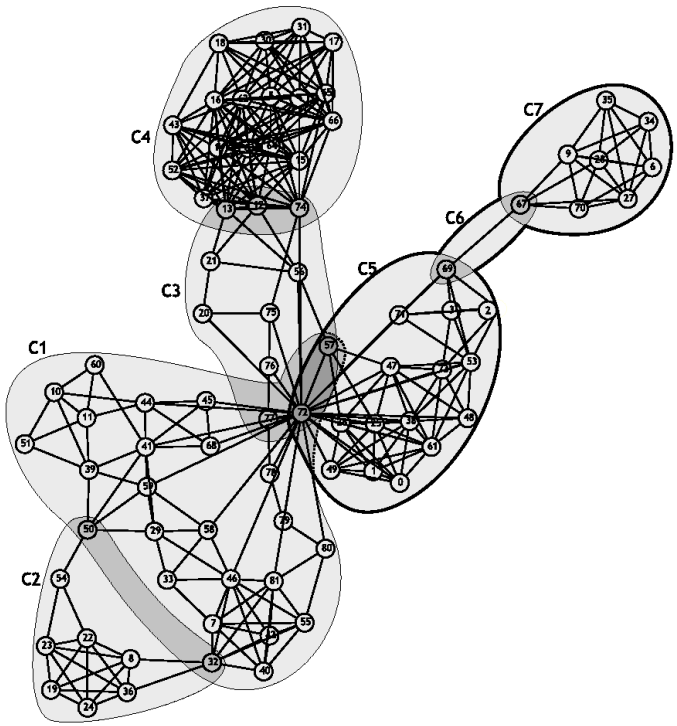
Un hypergraphe de contraintes



Propriétés

- Acyclicité (α-acyclicité [BFMY83])
- Problème polynomial
- Recouvrements acycliques

Exploitation de la structure



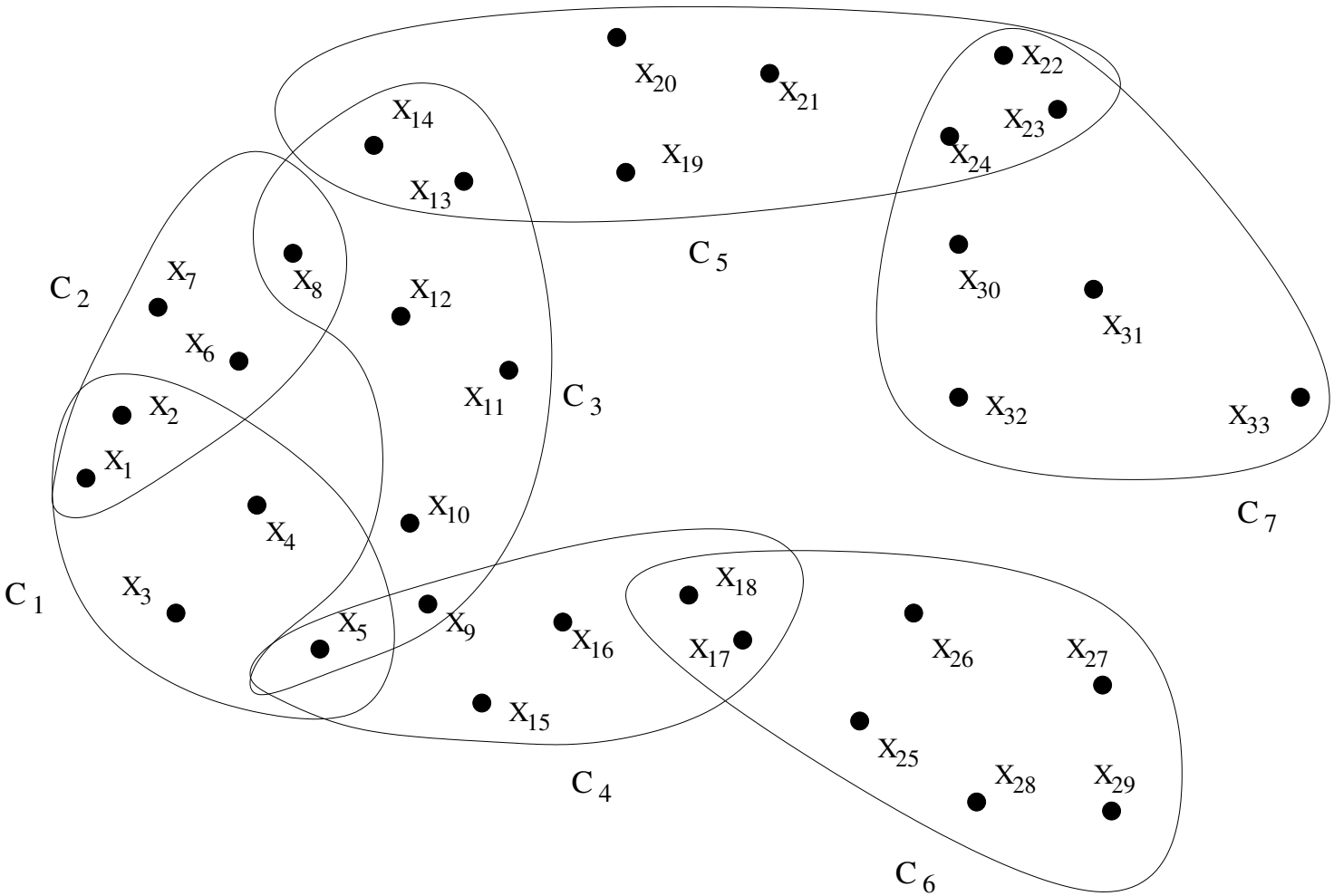
SCEN06 : n=100, d = 44, w=19

méthodes	espace recherche
BT	$d^n = 44^{100}$
BTD	$7 * d^{w+1} = 7 * 44^{20}$

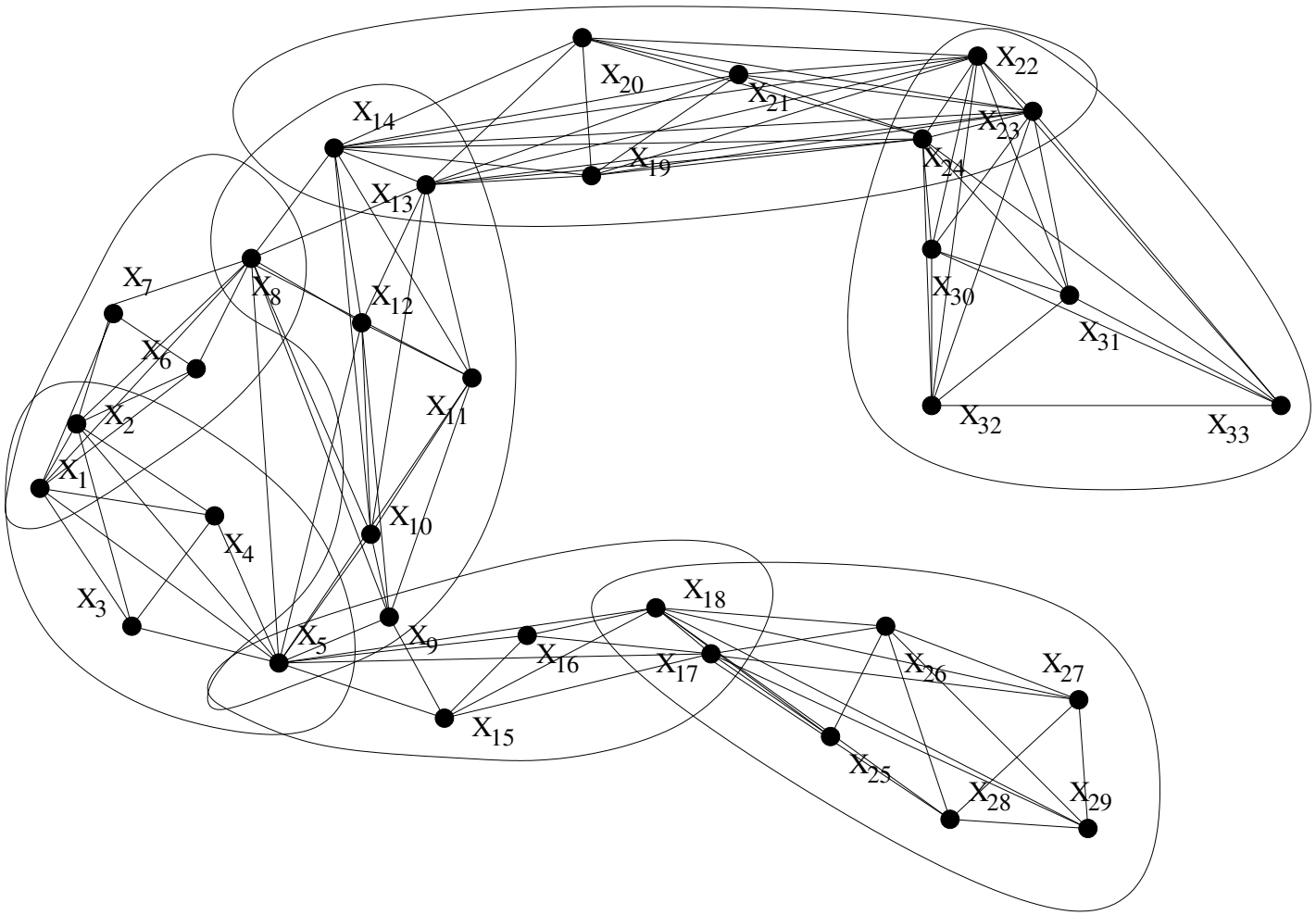
BTD : Backtracking on Tree-Decomposition [JT03]

- combine
  - la souplesse de l'énumération
  - les bornes de complexité des méthodes structurelles
- basée sur une décomposition arborescente de la 2-section (graphe primal) [Berge70] de l'hypergraphe de contraintes

2-section d'un hypergraphe



2-section d'un hypergraphe



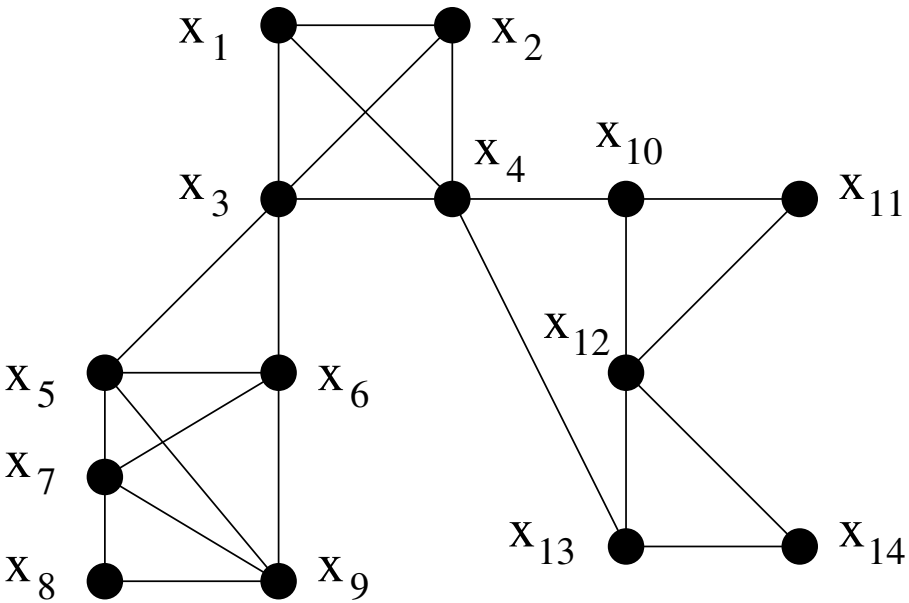
Décomposition arborescente

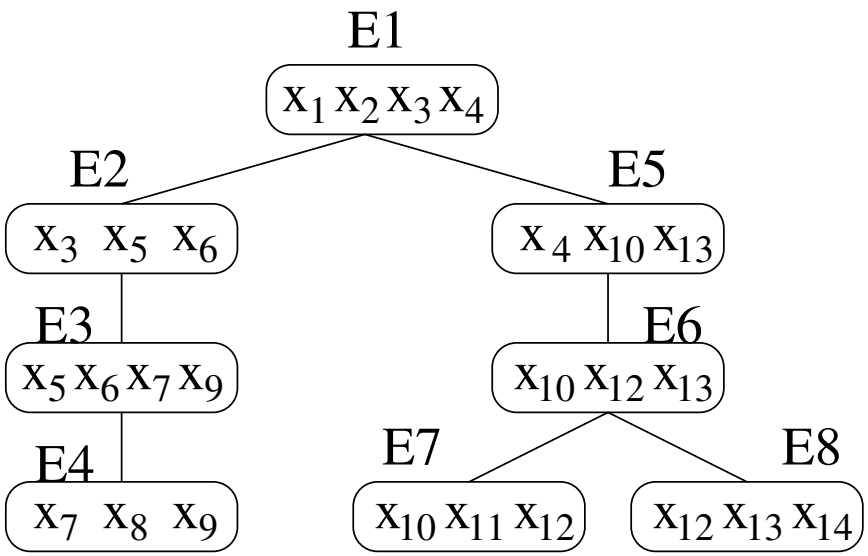
[RS86]

Etant donné un graphe $G = (X, C)$, une *décomposition arborescente* de G est un couple $(E, \mathcal{T})$ où $\mathcal{T} = (I, F)$ est un arbre avec I l'ensemble des nœuds et F celui des arêtes et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, qui vérifie :

- $\cup_{i \in I} E_i = X$,
- pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subset E_i$, et
- pour tout $i, j, k \in I$, si k est sur une chaîne allant de i à j dans $\mathcal{T}$, alors $E_i \cap E_j \subset E_k$.

Décomposition arborescente

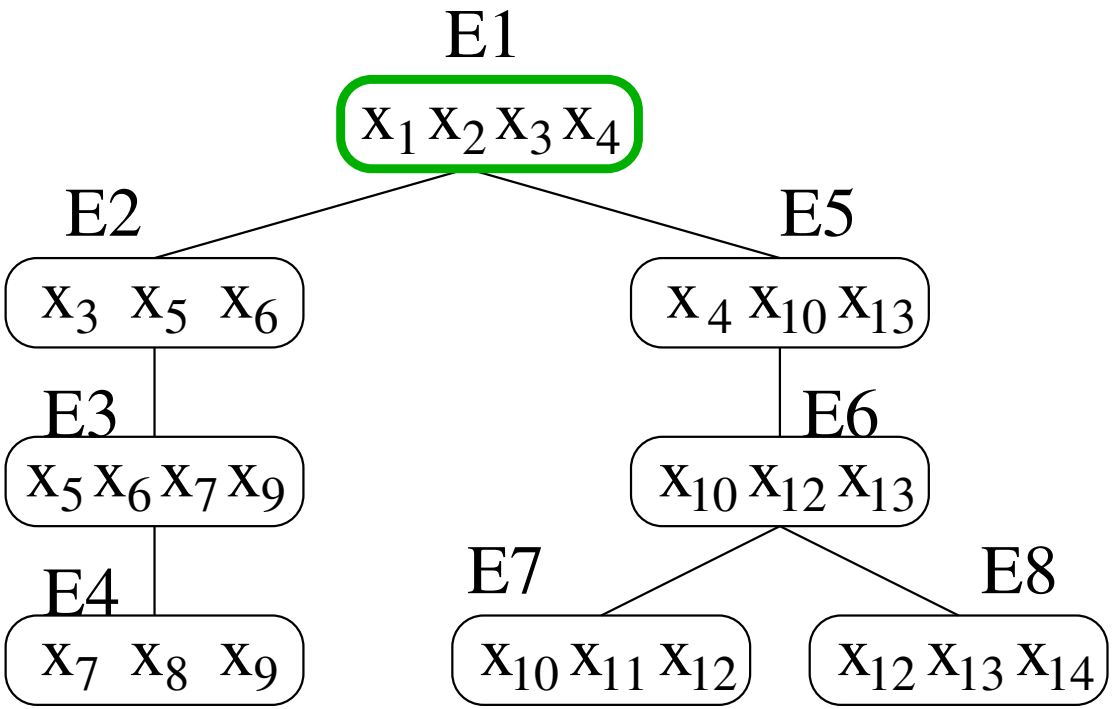




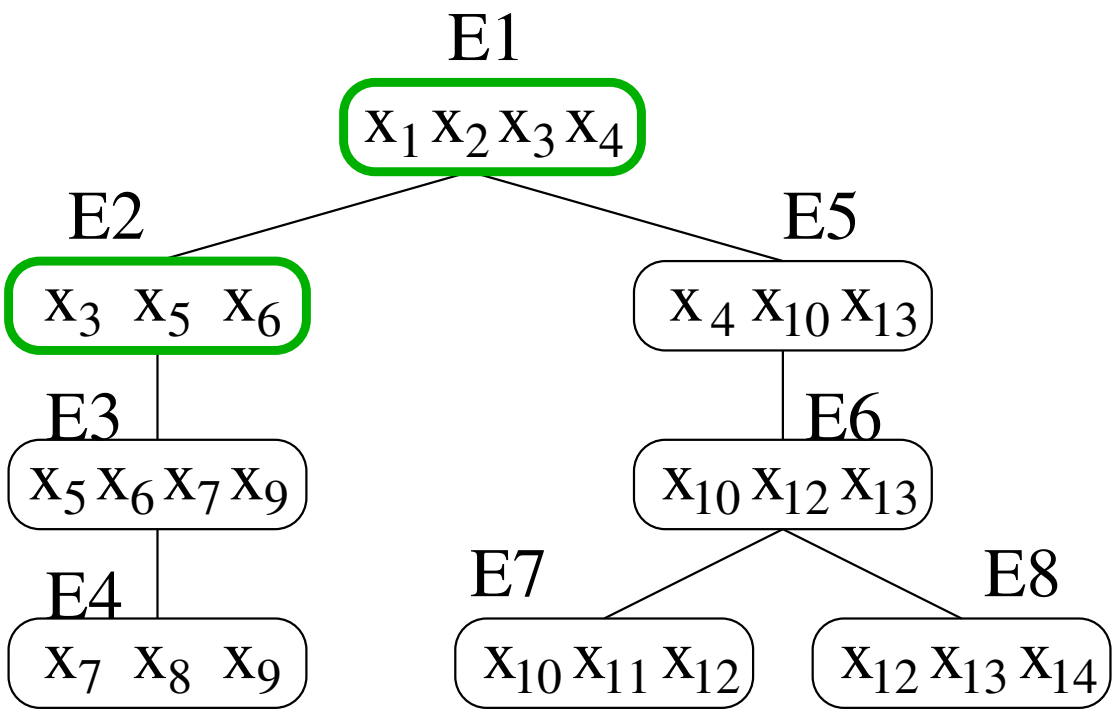
BTD

- Un ordre d'instanciation induit par la décomposition arborescente

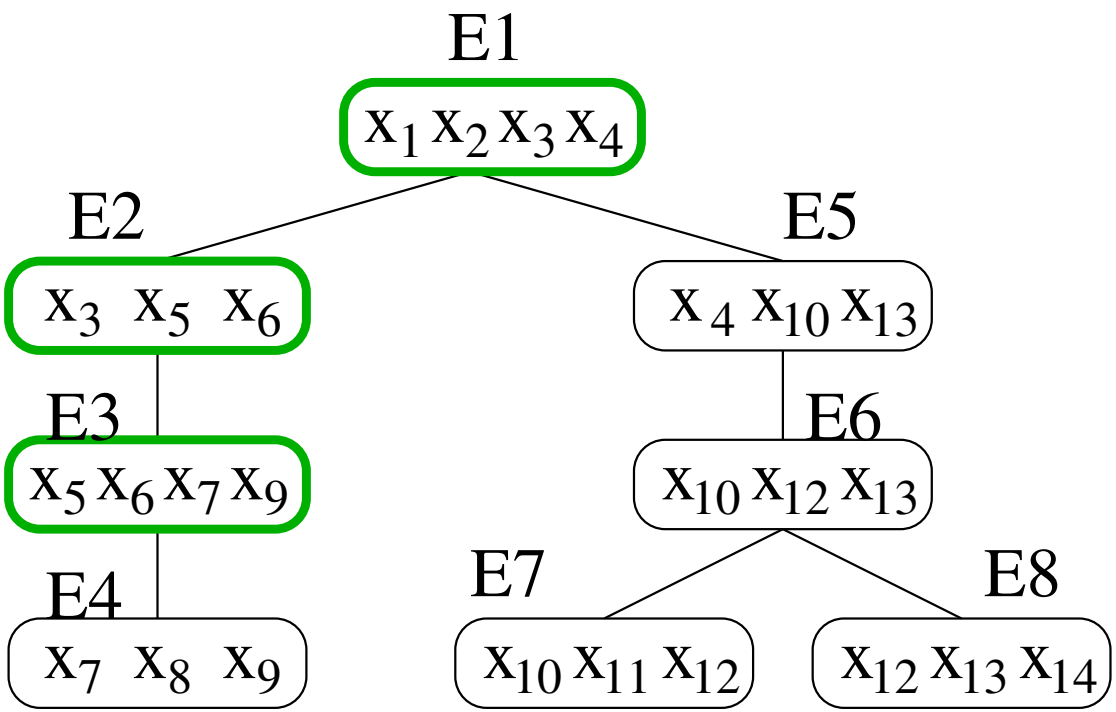
Ordre induit par la décomposition



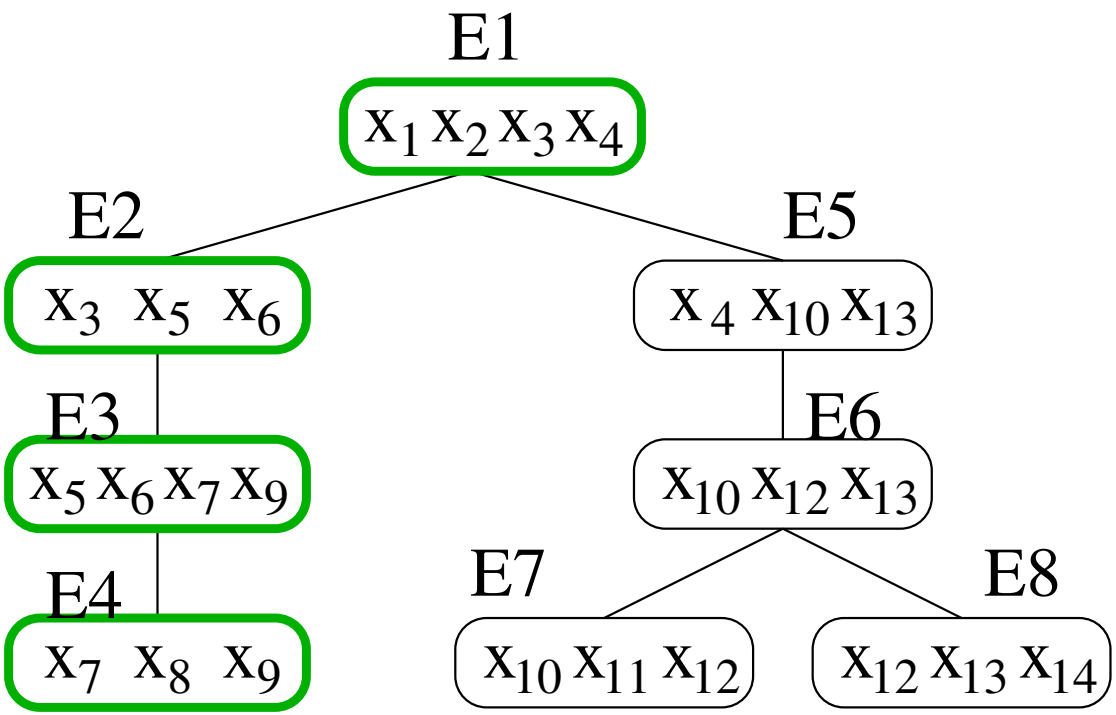
Ordre induit par la décomposition



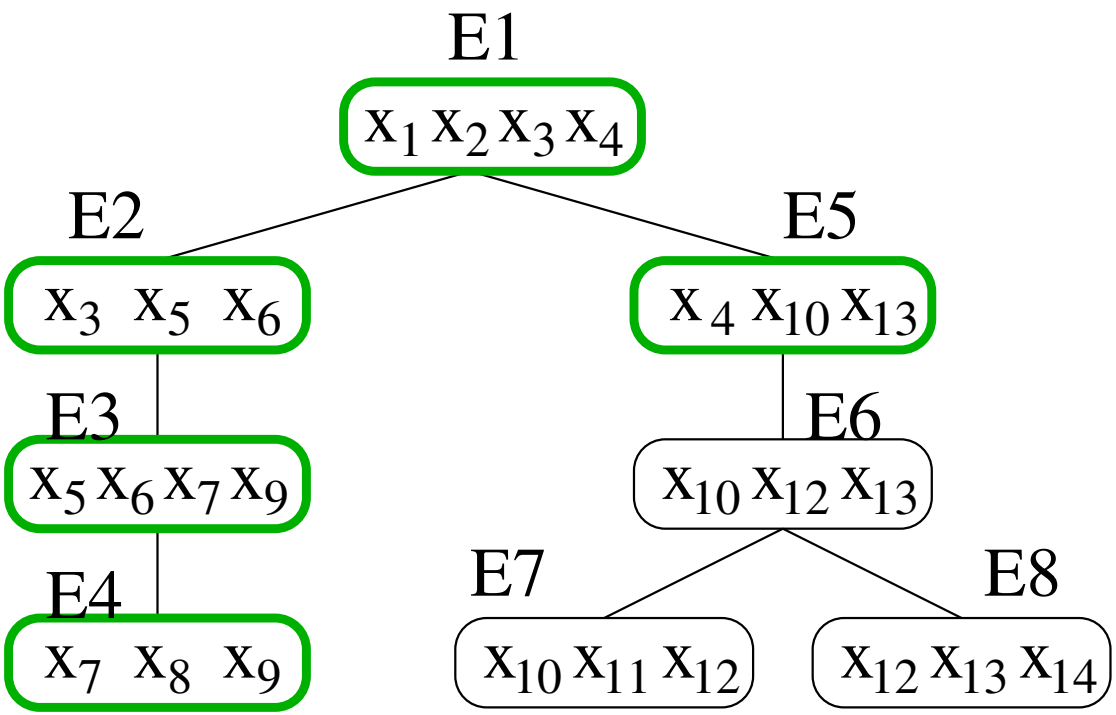
Ordre induit par la décomposition



Ordre induit par la décomposition



Ordre induit par la décomposition



BTD

- Un ordre d'instanciation induit par la décomposition arborescente
- Apprentissage de goods et nogoods structurels

Soient E_i un cluster et E_j l'un de ses fils. Un good (resp. nogood) structurel de E_i par rapport à E_j est une affectation consistante sur $E_i \cap E_j$ telle qu'il existe (resp. n'existe pas) d'extension consistante de cette affectation sur $Desc(E_j)$.

Goods structurels

$x_1 \leftarrow a$

$x_2 \leftarrow b$

$x_3 \leftarrow c$

$x_4 \leftarrow d$

$x_5 \leftarrow a$

$x_6 \leftarrow b$

$x_7 \leftarrow c$

$x_9 \leftarrow d$

$x_8 \leftarrow a$

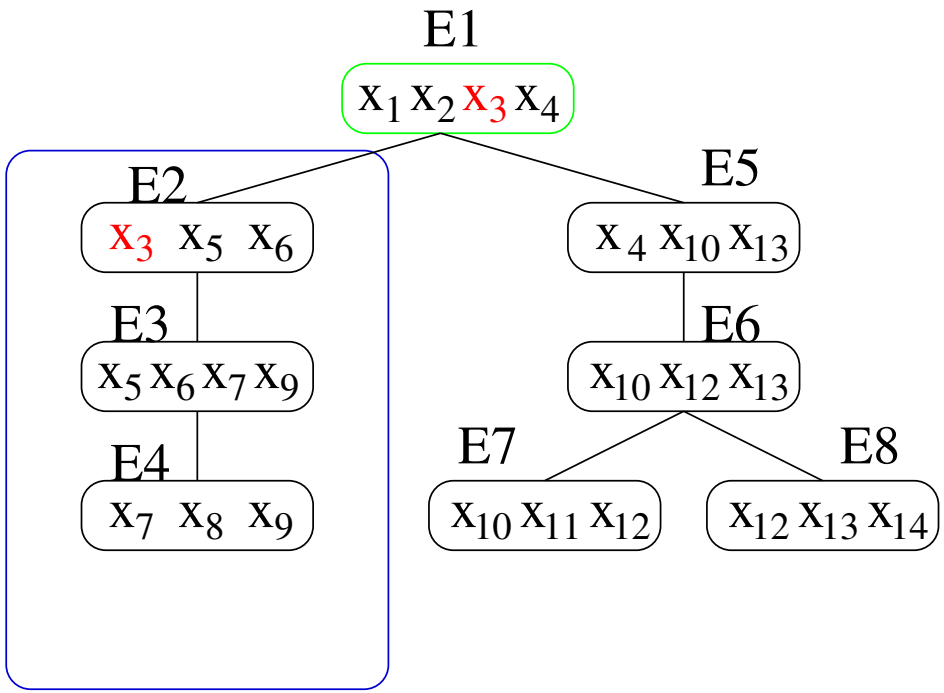
$x_{10} \leftarrow$

$x_{13} \leftarrow$

$x_{12} \leftarrow$

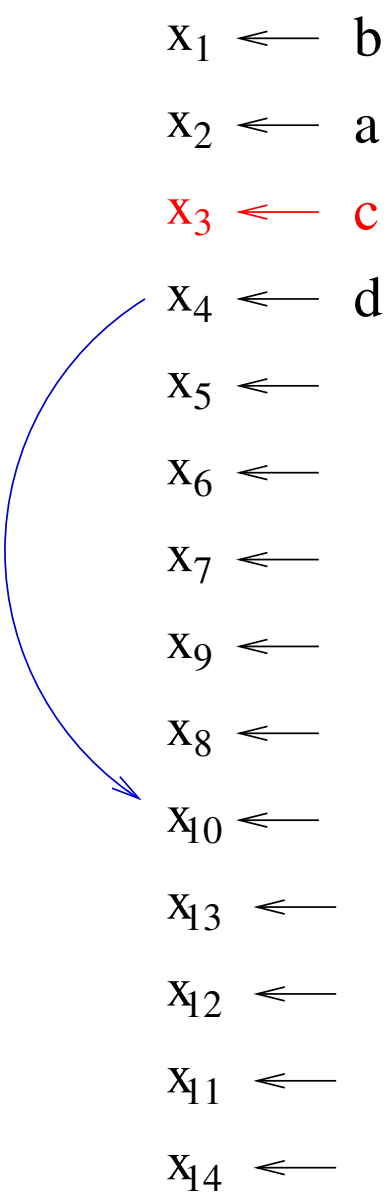
$x_{11} \leftarrow$

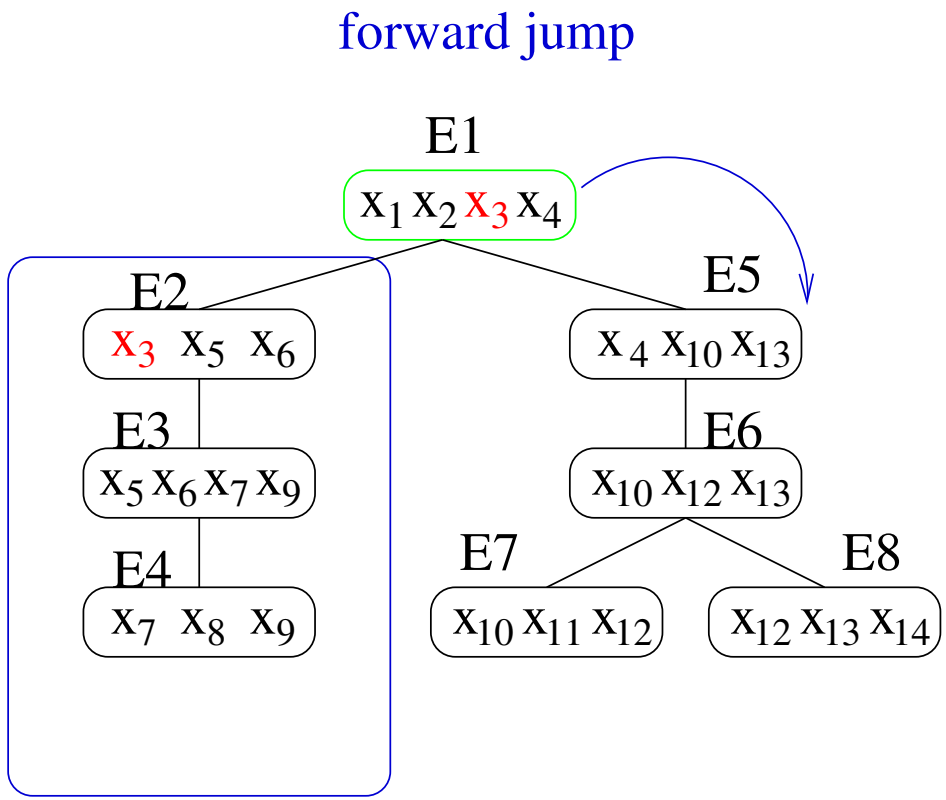
$x_{14} \leftarrow$



$x_3 \leftarrow c$: good

Exploitation des Goods structurels





$x_3 \leftarrow c$: good

Nogoods structurels

$x_1 \leftarrow c$

$x_2 \leftarrow a$

$x_3 \leftarrow d$

$x_4 \leftarrow b$

$x_5 \leftarrow$

$x_6 \leftarrow$

$x_7 \leftarrow$

$x_9 \leftarrow$

$x_8 \leftarrow$

$x_{10} \leftarrow$

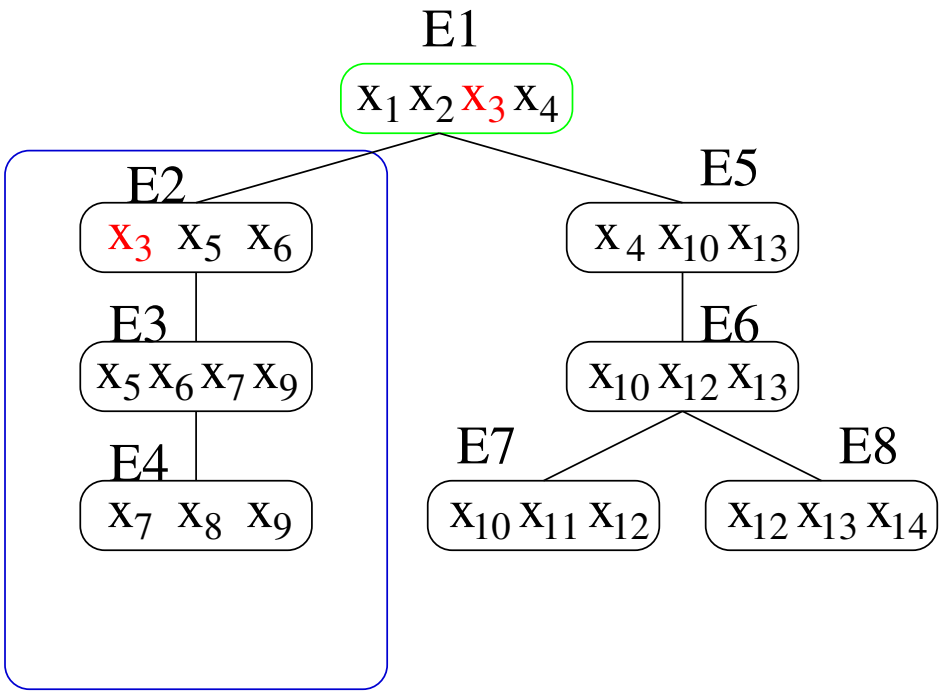
$x_{13} \leftarrow$

$x_{12} \leftarrow$

$x_{11} \leftarrow$

$x_{14} \leftarrow$

echec

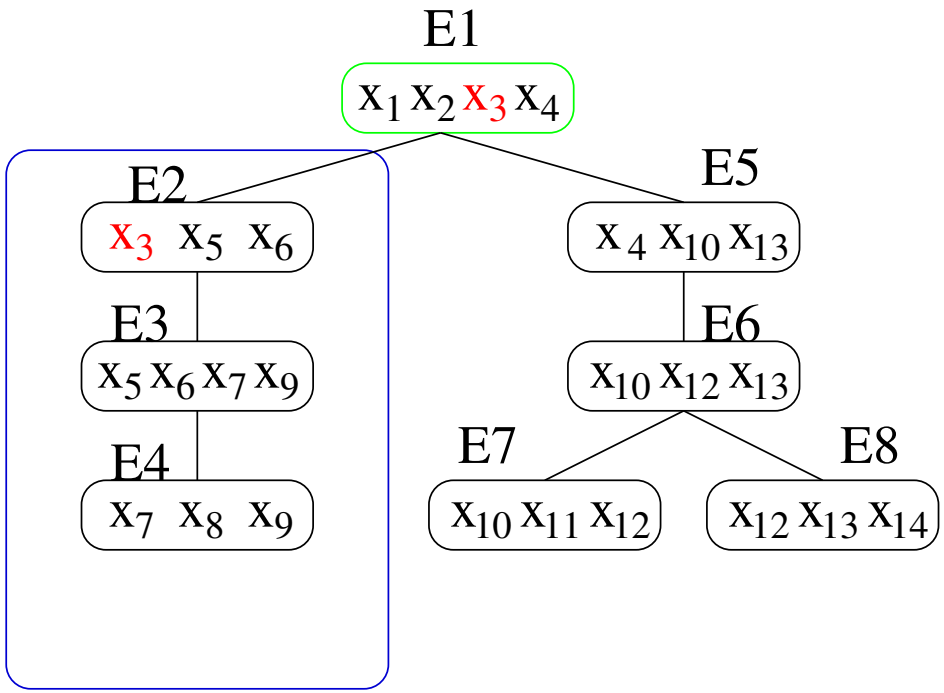


$x_3 \leftarrow d$: nogood

Exploitation des Nogoods structurels

$x_1 \leftarrow d$
$x_2 \leftarrow c$
$x_3 \leftarrow d$
$x_4 \leftarrow a$
$x_5 \leftarrow$
$x_6 \leftarrow$
$x_7 \leftarrow$
$x_9 \leftarrow$
$x_8 \leftarrow$
$x_{10} \leftarrow$
$x_{13} \leftarrow$
$x_{12} \leftarrow$
$x_{11} \leftarrow$
$x_{14} \leftarrow$

backtrack



$x_3 \leftarrow d$: nogood

Constraint Tractability Hierarchy

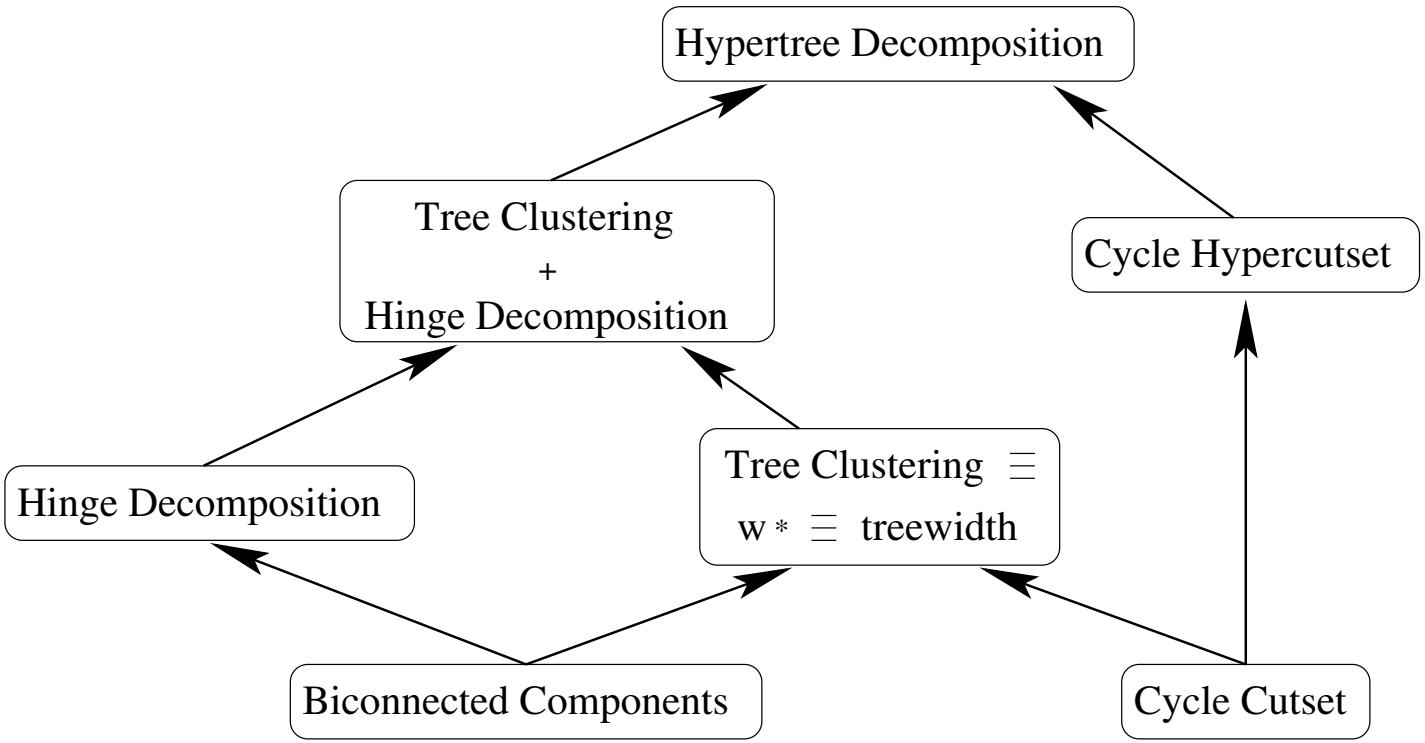


Figure – Hiérarchie des méthodes basées décomposition. Un arc (MD_1,MD_2) signifie que MD_2 généralise fortement MD_1.

HTD

- Méthode de résolution [GLS00]
- basée sur une Hypertree-Dec du problème

Décomposition en hyperarbre

[GLS00]

Etant donné un hypergraphe $H = (X, C)$, une *décomposition en hyperarbre, ou décomposition hyperarborescente* de H est un hyperarbre $HD = (T, \chi, \lambda)$ pour H qui satisfait les conditions suivantes :

(i) (T, χ) est une décomposition arborescente de H

(ii) pour chaque $p \in sommets(T), \chi(p) \subset \cup_{c \in \lambda(p)} c$,

(iii) pour chaque $p \in sommets(T), \cup_{c \in \lambda(p)} c \cap \chi(T_p) \subset \chi(p)$.

Hypertree-Dec

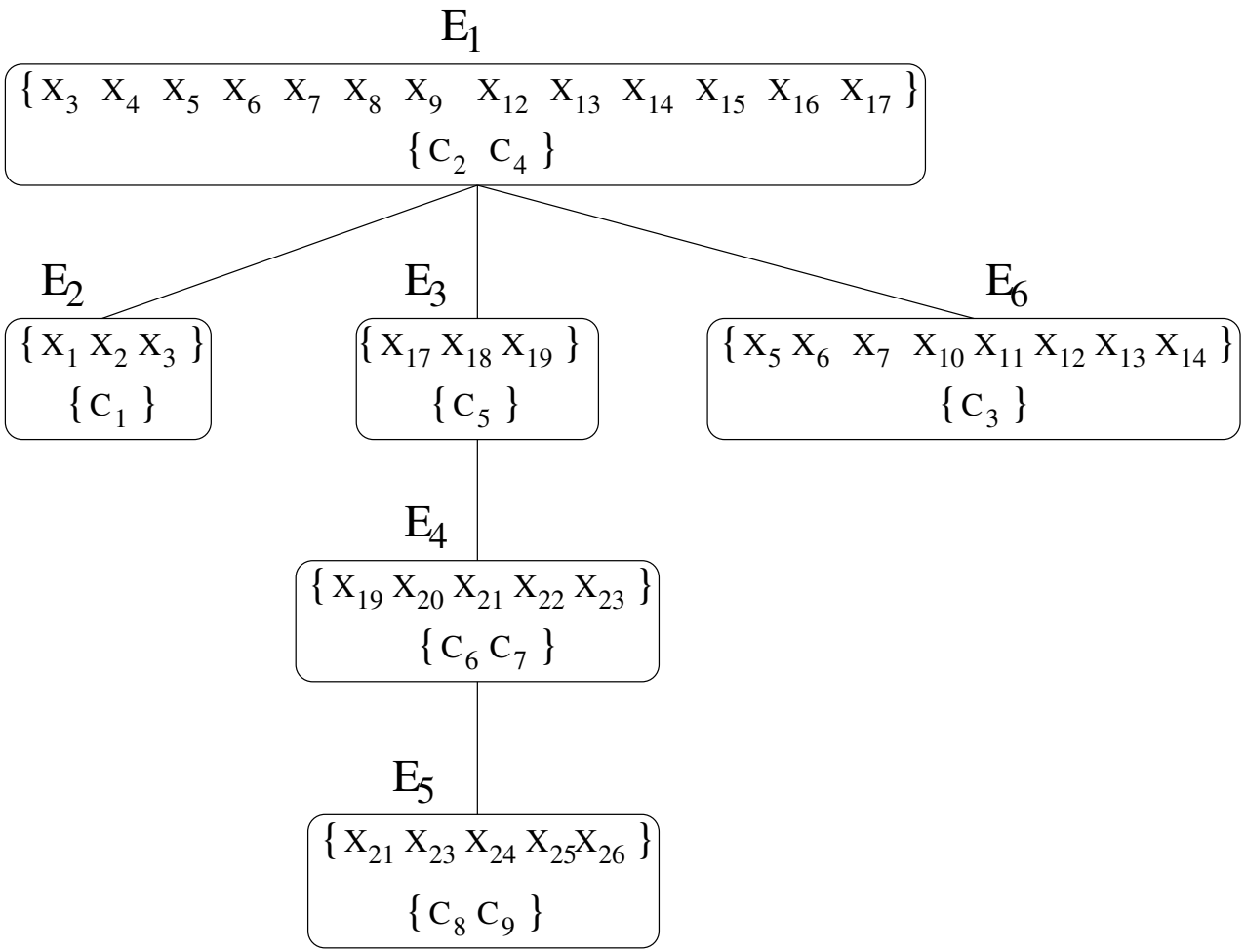


Figure – Une Hypertree-Dec optimale ($h = 2$). Complexité de HTD en $O(S^2)$