Logique

Problèmes de Satisfaction de Contraintes

Florent Madelaine et Mamadou Kanté

M2 Clermont

Logique et Graphes

Année 2013 - 2014

Plan du cours

Définitions des problèmes de contraintes

Complexité des problème de contraintes

Complexité pour un graphe des contraintes arborescent

Programmation dynamique

Le résultat le plus général

Définitions des problèmes de contraintes

Complexité des problème de contraintes

Complexité pour un graphe des contraintes arborescent

Programmation dynamique

Le résultat le plus général

Introduction

Le problème de satisfaction de contrainte est un problème combinatoire assez général de NP, qu'on peut voir à la fois comme une généralisation de SAT et comme une généralisation du problème de 3-colorabilité.

On parle plus généralement de problèmes de satisfaction de contraintes puisqu'on considère de nombreuses restrictions de ce problème général. Il existe deux méthodes principales : on restreint

- le graphe des contraintes, ou
- le langage des contraintes.

Dans ce cours nous allons présenter une brève introduction au premier type de restriction (on parle aussi de restriction structurelle).

Différentes définitions

Nous allons voir que les problèmes de contraintes peuvent se définir de plusieurs façons équivalentes.

- définition « classique » utilisée en intelligence artificielle
- définition combinatoire sous forme de problème d'homomorphisme
- définition logique comme problème de model checking du fragment positif primitif de la logique du premier ordre ($\{\exists, \wedge\}$-FO).

Définition IA « classique »

Une instance du problème de satisfaction de contraintes CSP est un triplet $(\text{Var}, \text{Dom}, \mathcal{C})$ où

- Var est un ensemble de variables,
- Dom est un ensemble de valeurs et
- $\mathcal{C}$ est un ensemble de contraintes :
  - chaque contrainte étant de la forme $(v_{i_1}, \ldots, v_{i_r}, R)$ où r est l'arité de la contrainte et $R \subseteq \text{Dom}^r$.

Une *solution* est une application des variables aux valeurs qui satisfait simultanément toutes les contraintes.

Un cas particulier important

Le cas booléen

SAT correspond à CSP avec un domaine booléen. Comme l'entrée est codée de manière différente (pas sous forme clausale), on parle de *Generalized Satisfiability*

Exemple

clause

$x \vee y \vee \bar{z}$

contrainte

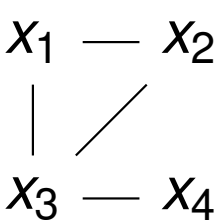
- (x, y, z)
- $\{0, 1\}^3 \setminus \{(0, 0, 1)\}$

Modéliser 3-col

Pour un graphe $G := (V, E)$ on a l'instance de CSP suivante

- variables Var $:= V$,
- valeurs Dom $:= \{1, 2, 3\}$ et
- contraintes $\mathcal{C} := \{(x_i, x_j, \neq) : \text{ pour chaque } (x_i, x_j) \in E\}$.

Exemple



- instance : Var $= \{x_1, x_2, x_3, x_4\}$, Dom $= \{1, 2, 3\}$ et $\mathcal{C} =$ $\{((x_1, x_2), \neq), ((x_1, x_3), \neq), ((x_3, x_2), \neq), ((x_3, x_4), \neq)\}$
- une solution : $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 1$

Deux structures sous-jacentes

Notre exemple

- instance : Var $= \{x_1, x_2, x_3, x_4\}$, Dom $= \{1, 2, 3\}$ et $\mathcal{C} = \{((x_1, x_2), \neq), ((x_1, x_3), \neq), ((x_3, x_2), \neq), ((x_3, x_4), \neq)\}$
- une solution : $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 1$

Deux structures

Variables $\mathcal{A}$ Valeurs $\mathcal{B}$

homomorphisme : $x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 3, x_4 \mapsto 1$

Homomorphisme

Vous avez dû rencontrer la notion d'homomorphisme en algèbre pour des groupes, des anneaux. De manière générale, il s'agit d'une application h (du domaine) d'une structure A vers (le domaine d')une structure B qui préserve les propriétés algèbriques de ces structures.
Dans le cas de structures relationelles, on a :

- pour tout symbole R, pour tout tuple $a_1, a_2, \ldots, a_r$ d'éléments de A, si $(a_1, a_2, \ldots, a_r) \in R^A$ alors $(h(a_1), h(a_2), \ldots, h(a_r)) \in R^B$

En particulier, pour le cas des graphes non orientés (une seule relation binaire qui est symmétrique) un homomorphisme envoie une arête du graphe A sur une arête du graphe B. Si les graphes sont orientés (une seule relation binaire) alors on envoie un arc sur un arc en préservant l'orientation.

CSP vu comme un problème d'homomorphisme

- instance : une paire de structures relationnelles similaires $(\mathcal{A}, \mathcal{B})$
- question : existe-t-il un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

où $\mathcal{A}$ représente la structure des contraintes et
$\mathcal{B}$ représente le language des contraintes

Exemple (3-col comme un homomorphisme dans K_3)

homomorphisme : $x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 3, x_4 \mapsto 1$

CSP comme problème de *model checking*

- instance : $\varphi_{\mathcal{A}}$ une formule de $\{\exists, \wedge\}$-*FO* et une structure $\mathcal{B}$
- question : $\mathcal{B} \models \varphi_{\mathcal{A}}$?

Exemple

$\varphi_{\mathcal{A}} = \exists x_1 \exists x_2 \exists x_3 \exists x_4 E(x_1, x_2) \wedge E(x_2, x_3) \wedge E(x_3, x_4) \wedge E(x_3, x_4)$

$\mathcal{B}$

Remarques

- Cette formulation permet de définir facilement des extensions de CSP comme QCSP.
- les formules de $\{\exists, \wedge\}$-*FO* sont connues sous le nom de requêtes conjonctives en BdD.

CSP vu comme un problème d'homomorphisme

- instance : une paire de structures relationnelles similaires $(\mathcal{A}, \mathcal{B})$
- question : existe-t-il un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

où $\mathcal{A}$ représente la structure des contraintes et
$\mathcal{B}$ représente le language des contraintes

Exemple (3-col comme un homomorphisme dans K_3)

homomorphisme : $x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 3, x_4 \mapsto 1$

Version non-uniforme du problème

On considère que B est fixée

- paramètre : $\mathcal{B}$
- instance : une structure relationnelle similaire $\mathcal{A}$
- question : existe-t-il un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

Notation pour ce problème de décision : CSP($\mathcal{B}$).

Exo

Donnez une formule MSO pour CSP($\mathcal{B}$) lorsque

- $\mathcal{B}$ est un triangle
- $\mathcal{B}$ est une clique à 4 sommets
- $\mathcal{B}$ est un cycle de 5 sommets
- $\mathcal{B}$ est un graphe quelconque

Complexité

Exo

Montrez que :

- Le problème général est NP-complet.
- Le problème est NP-complet pour un langage de contraintes comprenant une seule relation binaire et un domaine à 3 éléments.
- Le problème est NP-complet pour un langage de contrainte avec 4 relations ternaires et un domaine à 2 éléments.

Complexité

On conjecture que les problèmes de contraintes non-uniformes suivent une dichotomie : à savoir que chaque CSP non uniforme est soit facile (dans P) soit difficile (NP-complet). Cette conjecture, formulée par Feder et Vardi il y a 20 ans, est démontrée pour le cas de graphes non orientés [1] (Hell et Nešetřil'90) et pour des domaines booléens (Schaeffer'78) ou de taille 3 (Bulatov'03).

Pour les restrictions structurelles, on sait que le problème suit une dichotomie et que la notion importante sous-jacente est celle de largeur d'arborescence bornée du core du graphe des contraintes. Nous allons expliquer ce résultat aujourd'hui.

1. en utilisant la définition en terme d'homomorphisme

Définitions des problèmes de contraintes

Complexité des problème de contraintes

Complexité pour un graphe des contraintes arborescent

Programmation dynamique

Le résultat le plus général

CSP non-uniforme restreint

Soit $\mathscr{C}$ une classe de structures. On dénote par $CSP(\mathscr{C}, \mathcal{B})$ le problème suivant.

- paramètres : $\mathscr{C}$ et $\mathcal{B}$
- instance : une structure relationnelle $\mathcal{A}$ avec $\mathcal{A} \in \mathscr{C}$
- question : existe-t-il un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

Graphe des contraintes arborescent

Le graphe des contrainte est le graphe de Gaifman de la structure $\mathcal{A}$ (on dit aussi *primal graph*). Les sommets sont les variables ; et, deux sommets sont adjacents ssi les variables correspondantes interviennent simultanément dans une contrainte.

Nous avons vu que pour $\mathcal{B}$ fixé, le problème CSP($\mathcal{B}$) appartient à MSO. D'après le théorème de Courcelle, nous savons donc qu'il existe un algorithme linéaire (en $0(n)$) pour CSP($\mathcal{B}$) si pour toute instance $\mathcal{A}$, le graphe de Gaifman de $\mathcal{A}$ a largeur d'arborescence au plus k.

Graphe des contraintes arborescent

corollaire du théorème de Courcelle : pour tout langage de contrainte fixée, le problème de contrainte non-uniforme est facile à résoudre si le graphe des contraintes est arborescent.

slogan

CSP($\{\text{tw} \leq k\}$,$\mathcal{B}$) est polynomial pour tout langage de contrainte $\mathcal{B}$.

Deux défauts :

- les constantes de cet algorithme ne sont pas très bonnes. Par exemple pour la formule que nous avons utilisé pour la 3-colorabilité on aurait quelque chose de l'ordre de $0(2^{2^k} n)$
- l'algorithme est différent pour chaque structure $\mathcal{B}$! (de plus les algos ne s'uniformisent pas vraiment).

Une autre approche

Pour un problème spécifique de MSO, il est fréquent que

- le théorème de Courcelle ne soit en fait qu'une étape préliminaire permettant de confirmer qu'on a un algorithme polynomial ; et,
- que dans un second temps, on puisse élaborer un algorithme spécifique mais efficace utilisant le principe de programmation dynamique.

Pour le cas particulier des problèmes de contraintes uniformes, nous allons donner deux méthodes.

problèmes de contraintes uniformes

Soit $\mathscr{C}$ une classe de structures. On dénote par $CSP(\mathscr{C}, \_)$ le problème suivant.

- paramètre : $\mathscr{C}$
- instance : une paire de structures relationnelles similaires $(\mathcal{A}, \mathcal{B})$ avec $\mathcal{A} \in \mathscr{C}$
- question : existe-t-il un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

Un exemple

Nous allons voir un exemple due à Daniel Màrx (les transparents en anglais de ce cours sont les siens).

The Party Problem

PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

The Party Problem

PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

Do not invite a colleague and his direct boss at the same time!

The Party Problem

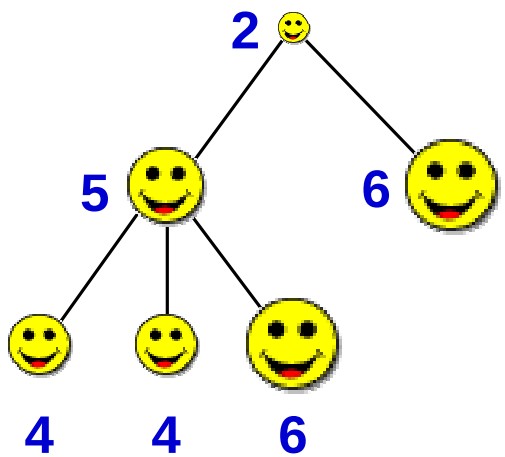
PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

Do not invite a colleague and his direct boss at the same time!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

The Party Problem

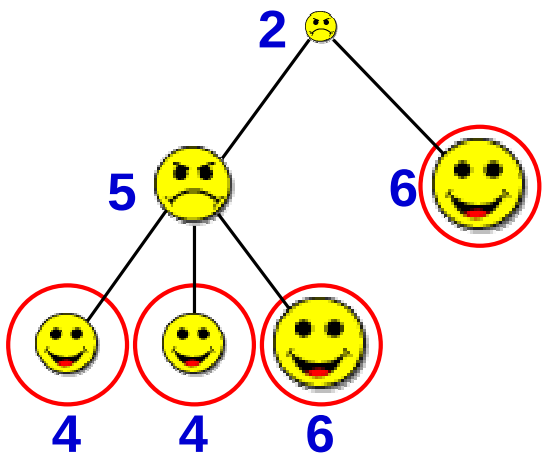
PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

Do not invite a colleague and his direct boss at the same time!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

Solving the Party Problem

Dynamic programming paradigm: We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Solving the Party Problem

Dynamic programming paradigm: We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Method:

Assume $v_1, \ldots, v_k$ are the children of v. Use the recurrence relations

$$B[v] = \sum_{i=1}^{k} A[v_i]$$
$$A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\}$$

The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).

Le cas des problèmes de contraintes

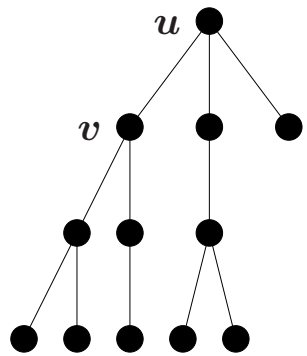
Il existe deux méthodes.

- La première est basée sur la prog dynamique et nécessite une décomposition du graphe des contraintes.
- La seconde utilise la k-cohérence et n'a pas besoin de la décomposition (mais elle ne résout que le problème de décision).

Nous allons expliquer rapidement les deux approches pour le cas de CSP binaires, c'est-à-dire où toutes les contraintes sont d'arité 2.

Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.

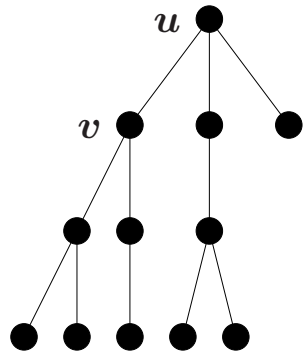


Proof 1: Dynamic programming. For $v \in V$, $d \in D$, let $x[v, d] =$ true if there is a partial solution on the subtree rooted at v such that variable v has value d.

The leaves are trivial. If the table is ready for the children of v, then computing $x[v, d]$ is easy.

Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.

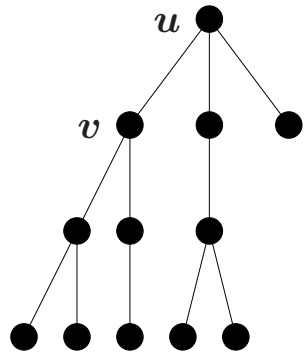


Proof 2: Arc consistency algorithm. If there is a constraint on (u, v) such that value d cannot appear on u, then remove every pair from every constraint on v that gives value d to u. Repeat.

Claim: When the algorithm stops, either every constraint is empty, or there is a solution.

Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.



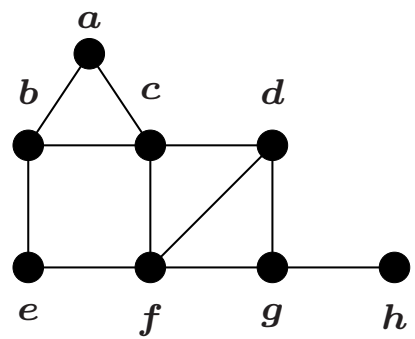
Can we generalize these ideas to wider classes of graphs?

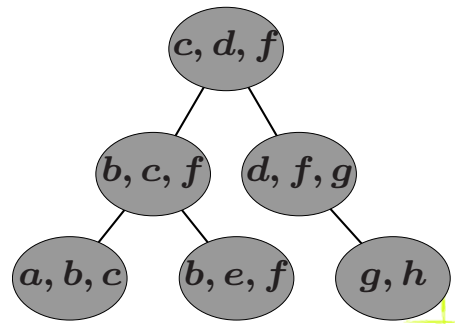
Treewidth

Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



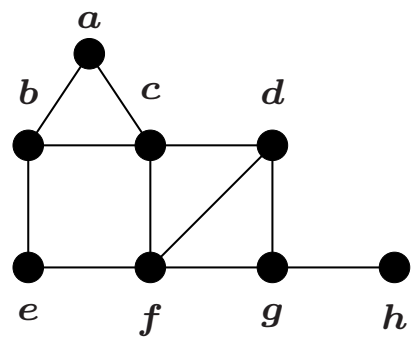


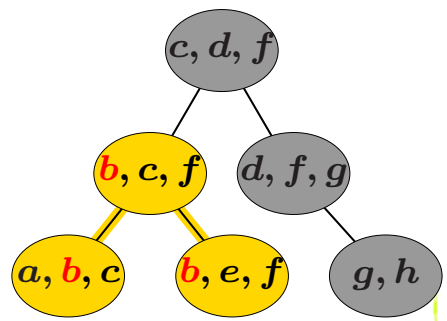
Treewidth

Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



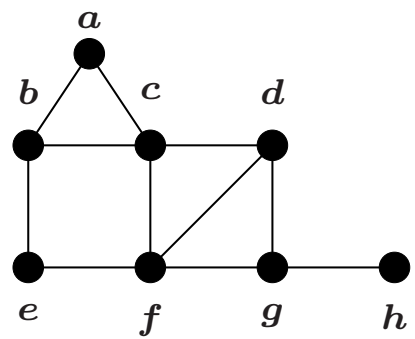


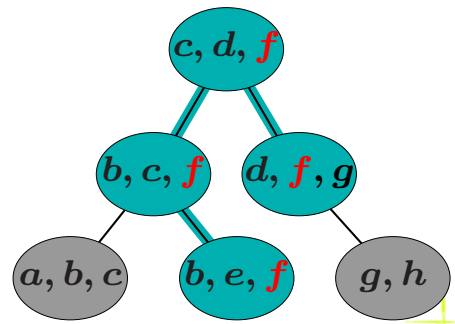
Treewidth

Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.





Treewidth and CSP

Fact: For every fixed k, CSP can be solved in polynomial time if the primal graph of the instance has treewidth at most k.

Two proofs:

- Using the k-consistency algorithm.
  Note: solves only the decision problem, does not give directly a solution.
- Using the tree decomposition.
  Note: requires a tree decomposition.

Consistency

A **partial solution** on $S \subseteq V$ is a mapping $f : S \to D$ that satisfies every constraint whose scope is in S.

Definition: An instance is **k-consistent** if for any subsets $X \subset Y \subseteq V$ of size at most $k+1$, every partial solution on X can be extended to Y.

Consistency

A **partial solution** on $S \subseteq V$ is a mapping $f : S \to D$ that satisfies every constraint whose scope is in S.

Definition: An instance is **k-consistent** if for any subsets $X \subset Y \subseteq V$ of size at most $k+1$, every partial solution on X can be extended to Y.

The k-Consistency algorithm generates a set of partial solutions that do no violate the consistency requirement.

k-Consistency

1. For every $S \subseteq V$ with $|S| \leq k+1$, generate the list L_S of all partial solutions on S.
2. If for some $X \subseteq Y$, there is an $f \in L_X$ having no extension in L_Y, then remove f and every extension of f from the lists.
3. Repeat Step 2 until there are no further changes.

Consistency

k-Consistency

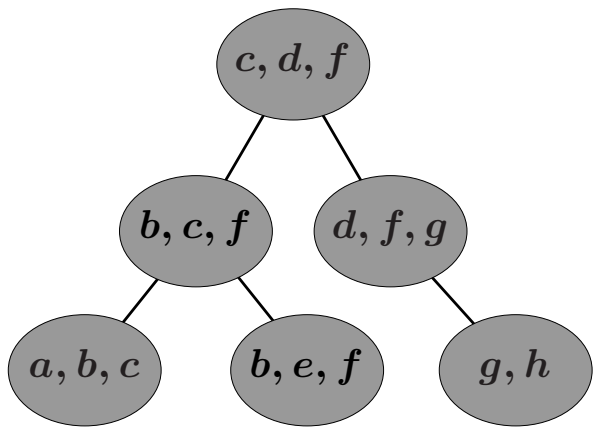
1. For every $S \subseteq V$ with $|S| \leq k+1$, generate the list L_S of all partial solutions on S.
2. If for some $X \subseteq Y$, there is an $f \in L_X$ having no extension in L_Y, then remove f and every extension of f from the lists.
3. Repeat Step 2 until there are no further changes.

Note:

- If an L_S is empty, then we can conclude that there is no solution.
- If $f \in L_Y$, then $f_{|X} \in L_X$ for every $X \subset Y$.
- The running time is polynomial for every fixed k: we manipulate subsets of size at most $k+1$ and the size of each L_S is at most $|D|^{k+1}$.

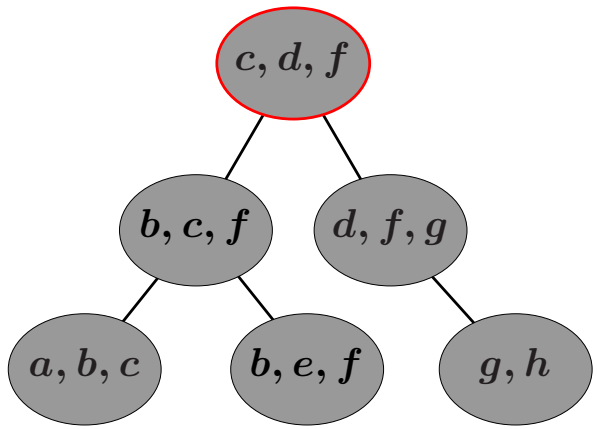
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



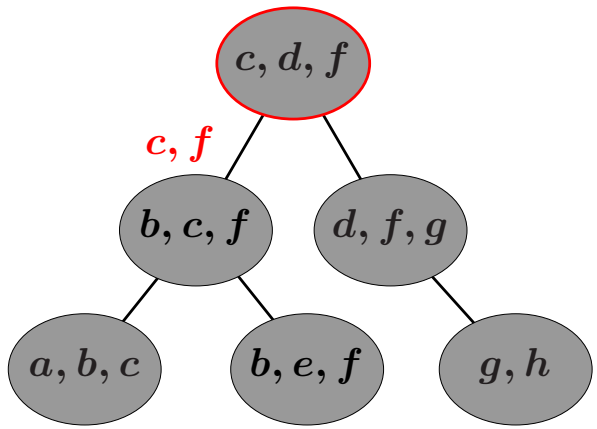
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



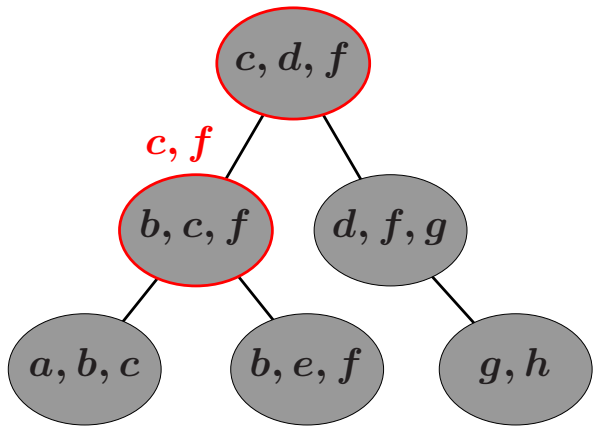
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



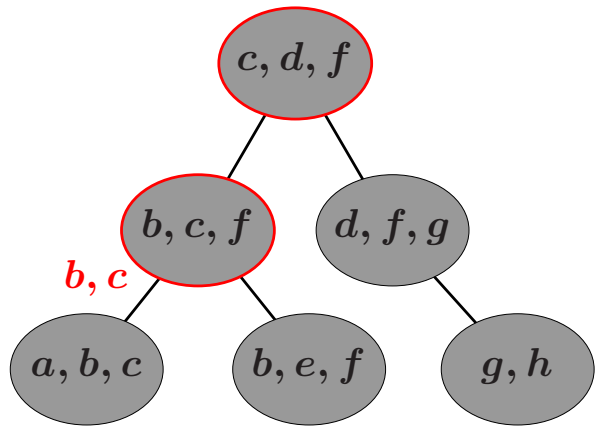
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



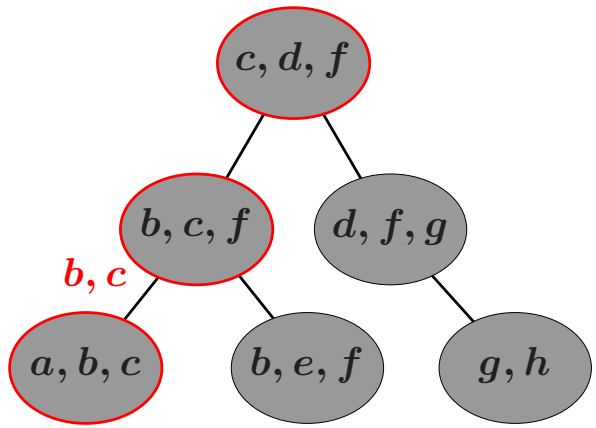
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



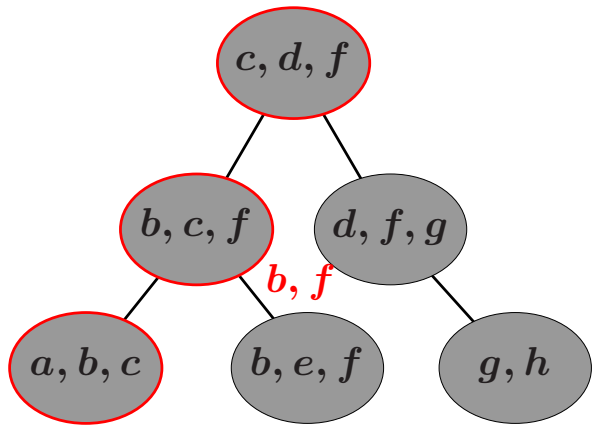
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



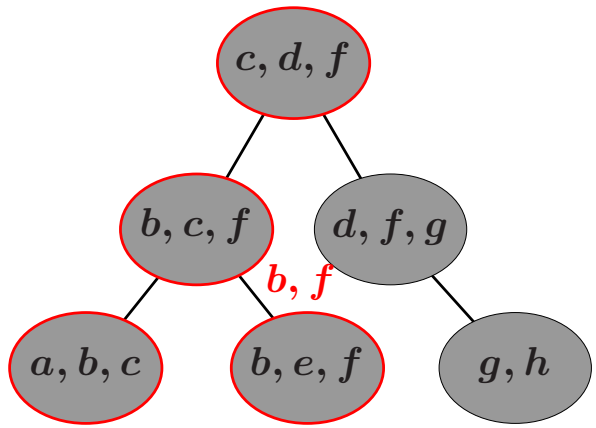
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



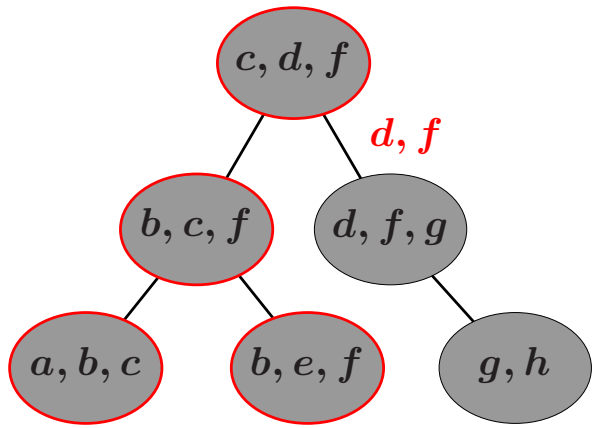
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



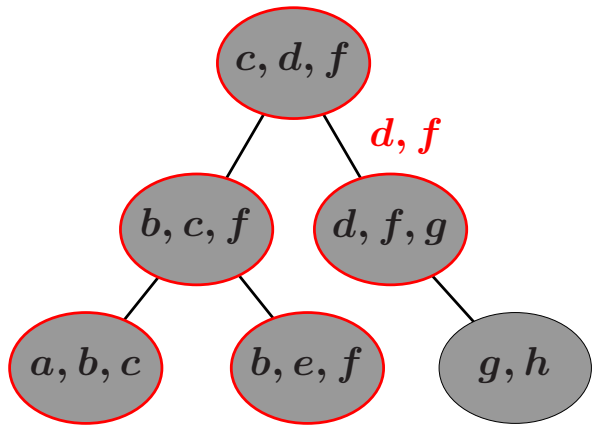
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



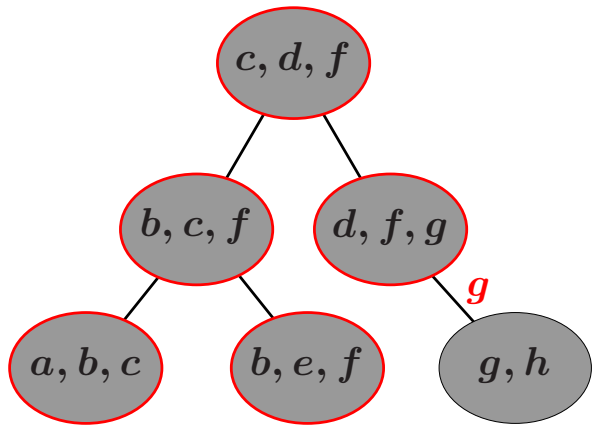
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



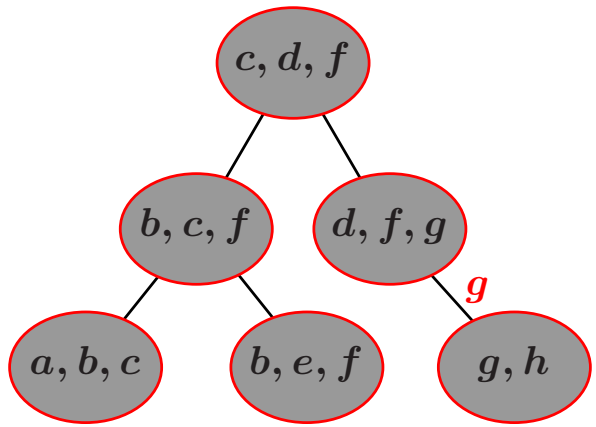
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



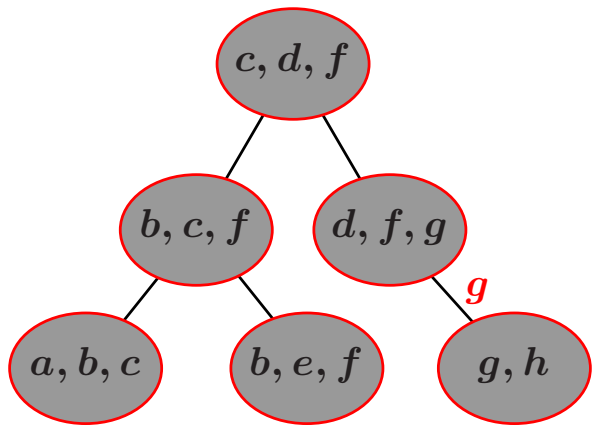
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



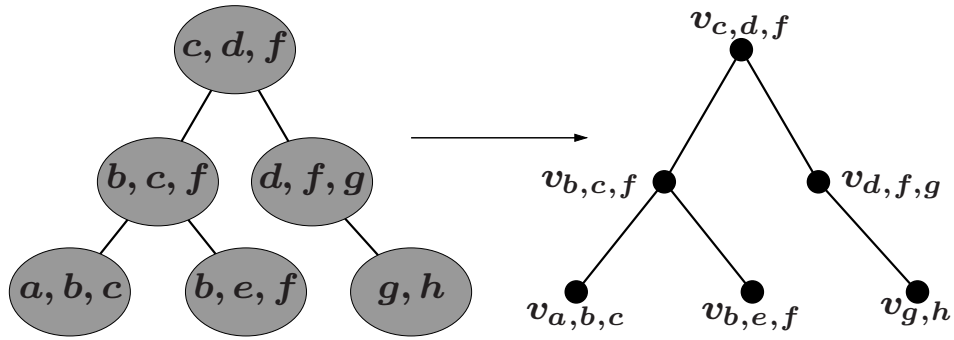
The properties of the tree decomposition ensure that

- each variable gets only a single value (connectedness property) and
- every constraint is satisfied (every clique appears in a bag).

Note: proof shows that a solution exists, but (unless we have a tree decomposition), does not show how to find one.

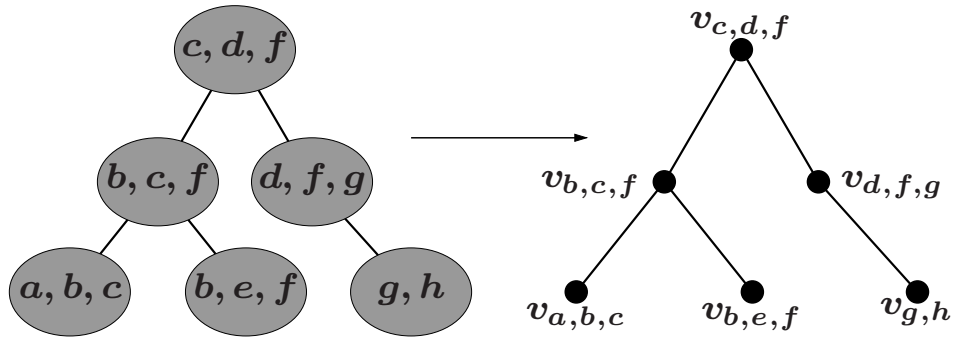
Solving CSP using a tree decompositions

Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



Solving CSP using a tree decompositions

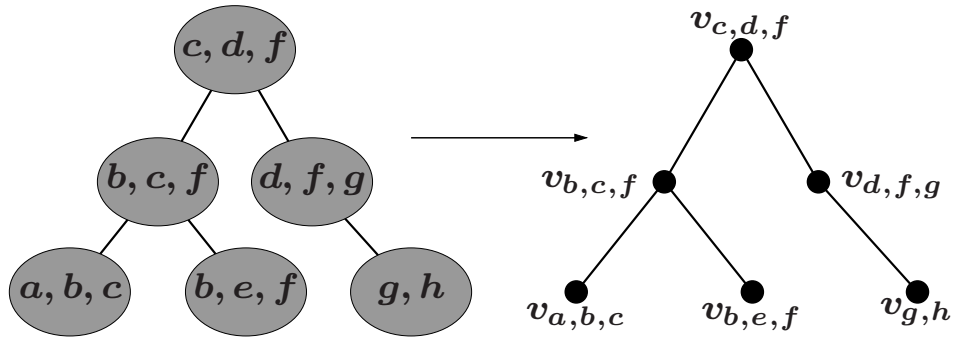
Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



The domain of variable $v_{a,b,c}$ is the set of all partial solutions on $\{a, b, c\}$. Binary constraint between $v_{a,b,c}$ and $v_{b,c,f}$ require that the two partial solutions agree on the intersection $\{b, c\}$.

Solving CSP using a tree decompositions

Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



The domain of variable $v_{a,b,c}$ is the set of all partial solutions on $\{a, b, c\}$. Binary constraint between $v_{a,b,c}$ and $v_{b,c,f}$ require that the two partial solutions agree on the intersection $\{b, c\}$.

- Instance has polynomial size for fixed k: domain size $\leq |D|^{k+1}$.
- There are no conflicts between the partial assignments.
- Every original constraint is satisfied by one of the partial solutions.

Même résultat au core près

On peut trouver une classe infinie de graphes dont la largeur d'arborescence n'est pas bornée mais pour lequel le CSP reste faisable (et même trivial comme on va le voir).

On considère la classe BIP des graphes bipartites. On a un homomorphisme d'un graphe bipartite quelconque vers un autre graphe $\mathcal{B}$ ssi $\mathcal{B}$ a une arête. Hors BIP comprend les grilles et vous avez sûrement vu que la classe des grilles n'est pas de largeur d'arborescence bornée.

En fait on peut montrer que le critère de faisabilité d'un CSP par rapport à des restrictions structurelles est le suivant : la classe des cores des graphes de contraintes est de largeur arborescente bornée.

Le core d'une structure

Le core $\mathcal{A}'$ d'une structure $\mathcal{A}$ est la [2] plus petit sous-structure induite qui est homomorphiquement équivalente à $\mathcal{A}$.
C'est-à-dire telle qu'il existe un homomorphisme de $\mathcal{A}$ dans $\mathcal{A}'$ et un homomorphisme de $\mathcal{A}'$ dans $\mathcal{A}$.

Cette notion est utile pour prouver des résultats autour des CSP. Dans un contexte plus concret, cette notion est très utile lorsqu'on se penche à l'intégration de données en Bases de données

2. on peut montrer que le core est unique à isomorphisme près

CSP et restriction structurelle

Théorème (Dalmau et. al. & Grohe et. al.)

CSP$(\mathscr{C}, \_)$ *est polynomial ssi* $\mathscr{C}'$ *a largueur arborescence bornée. où* $\mathscr{C}' := \{\mathcal{B}'$ *tel que* $\mathcal{A}'$ *est le core de* $\mathcal{A}$ *avec* $\mathcal{A} \in \mathscr{C}\}$

Au sujet de la démonstration

Pour la direction positive (dûe à Dalmau Kolaitis et Vardi), l'algorithme est basé sur la *k*-cohérence et la preuve repose sur une caractérisation par des jeux d'une logique à *k* variables associée. Il est important de noter que l'on n'a besoin ni du core de $\mathcal{A}$ ni de la décomposition arborescente de ce dernier (heureusement puisque l'un est l'autre ont des problèmes de décision associés qui sont NP-complet).

Pour la direction négative (dûe à Grohe), la preuve utilise comme hypothèse une conjecture de complexité paramétrée qui est l'analogue de P $\neq$ NP dans ce domaine.

Fin

1. Don’t hesitate to do the exercises online (so far one contribution only).
2. Last year’s exam is available on the platform.
3. You are welcome to send me scans/photos of solved exercises by email and I’ll try to respond within 1 to 3 days.