Résumé Partie II du Cours du M2 Decim
Complexité des CSP et des requêtes

Florent Madelaine

1 Motivation

Le problème de contraintes général est NP-complet puisqu'il permet de modéliser naturellement des problèmes difficiles bien connus comme Sat ou 3-Col. On cherche donc des restrictions du problème général pour lesquelles on dispose d'un algorithme efficace, c'est-à-dire qui s'exécute en temps polynomial. On parle donc des problèmes de contraintes. Ces problèmes de contraintes ont plusieurs définitions équivalentes.

- Définition IA classique « variables-valeurs-constraintes » ;
- Problème d'homomorphisme ;
- Inclusion de requêtes conjonctives ; et,
- Problème du *Model-checking* pour un fragment de FO[1].

De plus, même si on ne le verra pas dans ce cours, la *complexité* peut être caractérisée *algébriquement* dans certains cas. L'étude des problèmes de contraintes et de leur complexité touche ainsi à plusieurs disciplines de mathématiques et d'informatique : algèbre, bases de données, combinatoire, complexité, logique.

Figure 1 – Ubiquité des problèmes de contraintes.

Nous adoptons ici la définition en terme de problème d'homomorphisme. Vous avez dû rencontrer cette notion d'*homomorphisme* en algèbre pour des groupes, des anneaux, des corps *etc.* De manière générale, étant donné deux structures relationnelles $\mathcal{A}$ et $\mathcal{B}$ de même signature[2],

[1] FO = First Order, la logique du premier ordre.

il s'agit d'une application h (du domaine) d'une structure $\mathcal{A}$ vers (le domaine d')une structure $\mathcal{B}$ qui préserve les propriétés algébriques de ces structures. Dans le cas de structures relationnelles, l'application h doit vérifier que : pour tout symbole R d'arité r, pour tout tuple $a_1, a_2, \ldots, a_r$ d'éléments de A, si $(a_1, a_2, \ldots, a_r) \in R^{\mathcal{A}}$ alors $(h(a_1), h(a_2), \ldots, h(a_r)) \in R^{\mathcal{B}}$. En particulier, pour le cas des graphes non orientés (une seule relation binaire qui est symmétrique) un homomorphisme envoie une arête du graphe $\mathcal{A}$ sur une arête du graphe $\mathcal{B}$. Si les graphes sont orientés (une seule relation binaire) alors on envoie un arc sur un arc en préservant l'orientation.

CONSTRAINT SATISFACTION PROBLEM (CSP)
- **instance** : deux structures relationnelles de même signature $\mathcal{A}$ et $\mathcal{B}$.
- **question** : Est-ce-qu'il y a un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

Intuitivement, le domaine de $\mathcal{A}$ représente les variables de l'instance et celui de $\mathcal{B}$ les valeurs que ces variables peuvent prendre. Un tuple $(a_1, a_2, \ldots, a_r)$ de la relation R de la structure $\mathcal{A}$ (ce qu'on note $R^{\mathcal{A}}$) signifie que la contrainte R porte sur les variables $a_1, a_2, \ldots, a_r$. Un tuple de la relation correspondante $R^{\mathcal{B}}$ de la structure $\mathcal{B}$ liste une combinaison de valeurs autorisée par la contrainte R.

L'intérêt d'adopter la définition en terme d'homomorphisme est que les deux façons de restreindre le problème général apparaissent de manière naturelle. Les restrictions portent soit sur le *graphe des contraintes* (un graphe associé à la structure $\mathcal{A}$), soit sur le *langage des contraintes* (la structure $\mathcal{B}$). On parle de problème de contrainte uniforme dans le premier cas et de problème de contrainte non-uniforme dans le second cas. [3]

NON-UNIFORM CONSTRAINT SATISFACTION PROBLEM (CSP($\mathcal{B}$))
- **paramètre** : une structure relationnelle $\mathcal{B}$ (le *langage des contraintes*).
- **instance** : une structure relationnelle de même signature $\mathcal{A}$.
- **question** : Est-ce-qu'il y a un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

UNIFORM CONSTRAINT SATISFACTION PROBLEM (CSP($\mathscr{C}$,_))
- **paramètre** : une signature relationnelle fixée σ et une classe de structures relationnelles $\mathscr{C}$ ayant cette signature.
- **instance** : deux structures relationnelles de même signature $\mathcal{A}$ et $\mathcal{B}$, où $\mathcal{A}$ appartient à $\mathscr{C}$.
- **question** : Est-ce-qu'il y a un homomorphisme de $\mathcal{A}$ dans $\mathcal{B}$?

[2] C'est-à-dire que les noms et arités des symboles de relations des deux structures correspondent. Par exemple, la signature pour des graphes orientés aura un seul symbole E binaire permettant de coder les arcs.

[3] Cette terminologie est dû au fait que si on disposait d'algorithmes polynomiaux pour CSP($\mathcal{B}$) où $\mathcal{B}$ appartient à une famille de langage de contrainte $\mathscr{C}$, rien ne garantie *a priori* qu'on puisse disposer d'un algorithme uniforme pour CSP(_,$\mathscr{C}$) (qu'on définit de manière analogue à CSP($\mathscr{C}$,_)). Il y a d'une part le fait que $\mathcal{B}$ fait partie de l'entrée ce qui peut rendre l'algorithme dédié non polynomial, et d'autre part, même si c'était le cas, pour pouvoir utiliser l'algorithme dédié il faudrait être capable de détecter efficacement lorsqu'on peut l'employer. Dans ce second cas, on parle souvent de *méta-problème* : puis-je décider si je me trouve dans un cadre polynomial, et si oui quel algo efficace je dois choisir ?

Dans ce cours on n'abordera que les restrictions pour la partie gauche (uniform CSP). La complexité de ces problèmes est liée à des propriétés combinatoires, et une méthode concrète plus ou moins raisonnable de résolution consiste à s'appuyer sur des **décompositions arborescentes** du graphe des contraintes.

Pour les restrictions sur le langage des contraintes, ce sont des méthodes différentes utilisant de l'algèbre universelle qui interviennent.

2 Requêtes conjonctives, CSP, homomorphismes et coloriage

Définition IA « classique » Une instance du problème de satisfaction de contraintes CSP est un triplet $(\mathsf{Var}, \mathsf{Dom}, \mathcal{C})$ 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 $R \subseteq \mathsf{Dom}^r$.

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

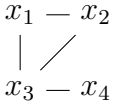
Un cas particulier important : le cas booléen. SAT correspond à CSP avec un domaine booléen. Comme l'entrée est codée de manière différente (pas sous forme clausale), on parle de *Generalized Satisfiability*

Exemple. La clause $x \vee y \vee \bar{z}$ correspond à la contrainte sur (x, y, z) autorisant tous les triplets sauf $\{(0, 0, 1)\}$, c'est-à-dire pour la relation booléenne d'arité 3 $\{0, 1\}^3 \setminus \{(0, 0, 1)\}$.

Modéliser 3-col Pour un graphe $G := (V, E)$ on a l'instance de CSP suivante

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

Exemple. Pour le graphe



- instance :

$\mathsf{Var} = \{x_1, x_2, x_3, x_4\}$,
$\mathsf{Dom} = \{1, 2, 3\}$ et
$\mathcal{C} = \{((x_1, x_2), \neq), ((x_1, x_3), \neq), ((x_3, x_2), \neq), ((x_3, x_4), \neq)\}$

- 1 solution : $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 1$

Deux structures sous-jacentes

Exemple. Revenons sur notre exemple, on peut faire apparaître 2 structures.

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

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 algébriques de ces structures.

Dans le cas de structures relationnelles, 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.

On retombe donc sur la définition annoncée en introduction.

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

Requêtes conjonctives Les requêtes SQL sont majoritairement des requêtes conjonctives (select des colonnes from des tables where des conditions de jointures posant des égalités entres certaines colonnes). Ce sont des requêtes qu'on peut écrire en logique du premier ordre sous la forme suivante :

$$\exists \text{ des variables } \bigwedge \text{ des faits sont vrais}$$

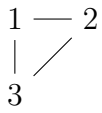
C'est-à-dire qu'on a des formules utilisant seulement le quantificateur $\exists$ et le connecteur $\wedge$ (quand il n'y a pas de quantificateur universel ni de disjonction ni de négation, on peut facilement enlever les égalités en renommant des variables).

Définition évaluation de requêtes

- entrée : $\varphi_{\mathcal{A}}$ une requête conjonctive et $\mathcal{B}$ une structure relationnelle (une base de données)
- question : est-ce-que $\mathcal{B} \models \varphi_{\mathcal{A}}$?

Exemple. Si on revient à notre exemple de 3 coloration, la base de données $\mathcal{B}$ est le triangle qu'on a déjà vu décrivant les couleurs autorisées À la structure $\mathcal{A}$ on associe une requête conjonctive canonique qui liste explicitement tous les faits de $\mathcal{A}$.

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



Bases de données $\mathcal{B}$

Une dernière façon de voir le problème est avec 2 requêtes conjonctives.

Inclusion de requêtes conjonctives (*Conjunctive Query Containment*)
- entrée : $\varphi_{\mathcal{A}}$ et $\varphi_{\mathcal{B}}$ deux requêtes existentielles positives conjonctives.
- question : est-ce-que $\varphi_{\mathcal{A}} \subseteq \varphi_{\mathcal{B}}$?

Ici par inclusion des requêtes conjonctives $\phi \subseteq \psi$, on veut dire que pour n'importe quelle base de données $\mathcal{D}$, si $\mathcal{D}$ satisfait ϕ alors $\mathcal{D}$ satisfait ψ.

Théorème 1 (Chandra, Merlin '77)**.** *Soit $\varphi_{\mathcal{A}}$ et $\varphi_{\mathcal{B}}$ les formules existentielles positives conjonctives canoniques associées à $\mathcal{A}$ et $\mathcal{B}$. Alors, $\mathcal{A} \to \mathcal{B}$ ssi $\varphi_{\mathcal{A}} \subseteq \varphi_{\mathcal{B}}$ ssi $\mathcal{B} \models \varphi_{\mathcal{A}}$.*

En conclusion, tous ces problèmes sont fondamentalement les mêmes sauf que concrètement les tailles des objets et donc les paramétrisation qu'on peut considérer raisonablement sont différentes.

Ainsi en bases de données, on a souvent une très grosse base $\varphi_{\mathcal{B}}$ et comparativement une toute petite requête $\varphi_{\mathcal{A}}$.

Pour des contraintes, on a plutôt pleins de petites contraintes (petit $\mathcal{B}$) et beaucoup de variables (gros $\mathcal{A}$).

Pour les contraintes, il faudrait même considérer des hypergraphes puisque les contraintes ne sont pas forcément d'arité borné (exemple : contraintes `alldif`).

Malgré tout, dans la suite on se concentrera sur les problèmes de contraintes et l'archétype qu'est le problème de coloriage puisque pour l'approche qui nous intéresse ici (graphe des contraintes qui ressemblent à un arbre) ces problèmes capturent une certaine essence de ce qu'on ferait sur des requêtes tout en simplifiant un peu le problème étudié.

3 Digression : le problème de la fête

Un petit exemple pour commencer

Problème de la fête au travail (Party Problem)
- **But** : Inviter des collègues pour une fête.
- **Maximiser** : le *fun factor*
- **Contrainte** : Tout le monde doit s'amuser (ne pas inviter le patron direct d'un collègue !)

En pratique, l'instance est un arbre avec des poids sur les noeuds (1 noeud = 1 collègue, son poids = son *fun factor* et son père = son chef direct). La sortie est un ensemble indépendant de poids maximal (*fun factor* de la fête).

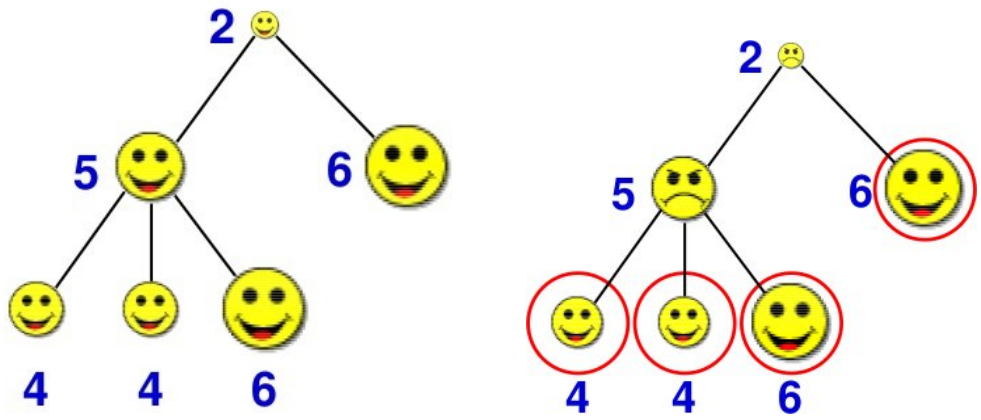


Figure 2 – Instance et Solution du Party Problem

Exercice 1. Proposer une méthode pour résoudre le problème. Indications.
- utiliser une méthode *bottom-up* ; et,
- du point de vue d'un sous-arbre de racine v, soit v appartient à la solution qu'on veut construire, soit v n'appartient pas à cette solution.

Notre but. Le problème de la fête s'appelle (pour des graphes) Maximum weight Independent set. C'est une généralisation de Independent Set qui est NP-complet. Or sur un arbre on vient de voir que ce problème devient polynomial et même *linéaire*. Le problème ne devient pas par contre complètement trivial, et on peut espérer que :
- l'approche fonctionne pour d'autres problèmes NP-complets (coloration, vertex cover, ...)
- l'approche fonctionne pour des arbres qui ne sont pas des arbres, mais peuvent être décrits par une structure de donnée arborescente (notion de largeur arborescente).

4 Rappels : définition et exemples décomposition arborescente

La notion de décomposition arborescente est apparue en théorie des graphes, au début des années 70. Cette notion a des applications algorithmiques au niveau théorique (via des résultats très génériques utilisant de la logique) mais aussi au niveau pratique via des méthodes plutôt efficaces qui utilisent des techniques de *programmation dynamique*).

Définition informelle

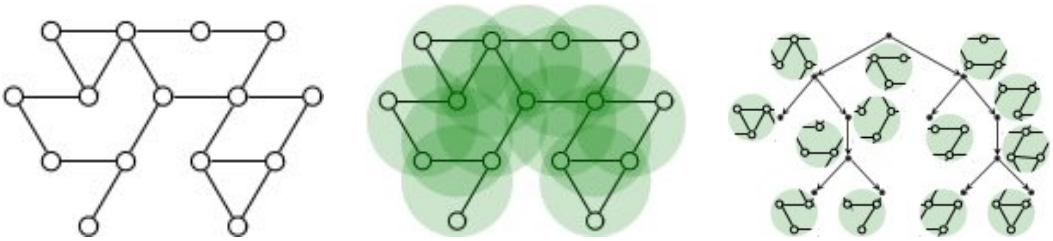


FIGURE 3 – Graphe, recouvrement par des sacs, arbre de sacs.

- On recouvre le graphe par des petits sacs.
- Ces sacs sont les noeuds d'un arbre
- Toutes les arêtes sont dans un sac

(voir Figure ci-dessus)

- Pour n'importe quel sac x, les sacs contenant x forment un sous-arbre.

(voir ci-dessous)

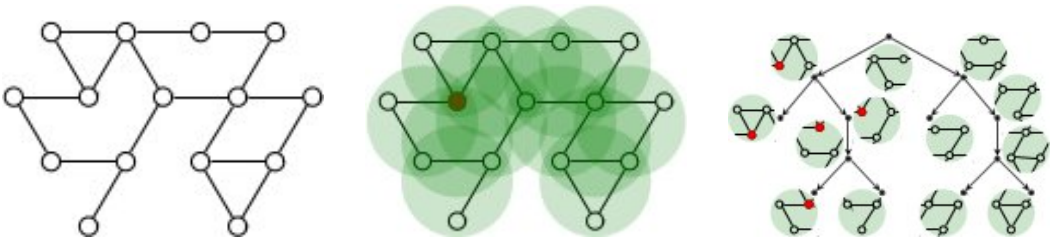


FIGURE 4 – 1 sommet correspond à un sous-arbre.

Définition formelle

Une *décomposition arborescente* d'un graphe G est un arbre $\mathcal{T}$ dont les sommets sont des sous-ensembles de sommets de G, qu'on appellera *sacs*, tel que :

- Tout sommet du graphe G appartient à un sac
- Toute arête du graphe G appartient à un sac
- Pour tout sommet x, l'ensemble des sacs contenant x est un sous-arbre de $\mathcal{T}$

La largeur de la décomposition est égal au nombre d"éléments du plus grand sac *moins 1*[4].

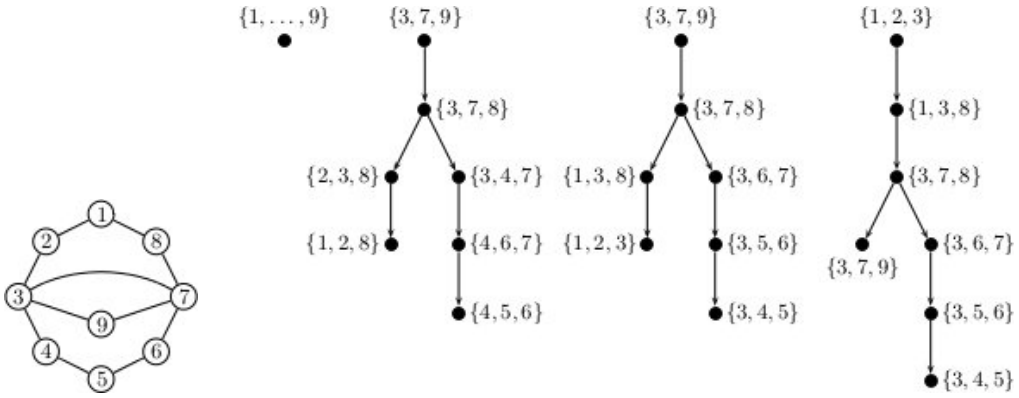


FIGURE 5 – Un graphe peut avoir plusieurs décompositions.

Exercice 2. Montrer que pour un arbre T, on peut toujours trouver une décomposition de largeur 1.

Exercice 3. Donner une décomposition arborescente de C_5 de largeur 3.

Distance avec un arbre

La *largeur d'arborescence* d'un graphe G est la largeur minimale d'une décomposition arborescente de ce graphe, qu'on notera $\mathrm{tw}(G)$ (en anglais, on dit *the treewidth of* G).

C'est une mesure de la distance à un arbre. Plus le nombre est petit, plus le graphe ressemble à un arbre.

Exercice 4. Quel est la largeur d'arborescence d'un arbre ? d'un cycle à n sommets, $n \geq 3$? d'une clique K_n, $n \geq 2$?

5 Approfondissement : décomposition arborescente

Lemme 2. *Soient G et H deux graphes. Si G est un mineur de H (c'est-à-dire qu'on peut l'obtenir par une séquence de contractions d'arêtes, suppressions d'arêtes et suppressions de sommets) alors* $\mathrm{tw}(G) \leq \mathrm{tw}(H)$.

Exercice 5. Démontrer le lemme ci-dessus. Indication : faites une preuve par récurrence sur le nombre d'opérations élémentaires (oubli sommet, oubli arête ou contraction d'arête) nécessaires pour construire le mineur ; et, transformez une décomposition du graphe avant l'opération en une décomposition du nouveau graphe.

[4]C'est une convention pour que les arbres aient une décomposition de largeur 1.

Exercice 6. Montrer qu'on peut toujours supposer que si G a largeur arborescente k, alors on peut toujours supposer que G admet une *petite décomposition*, c'est-à-dire une décomposition dont tous les sacs sont 2 à 2 incomparables pour l'inclusion.

On comprend maintenant un peu mieux quels graphes sont proches d'arbres. La question est lesquels sont très différents des arbres. On verra que les grilles sont un exemple canonique.

Exercice 7. Montrer que un graphe de grille de n lignes et n colonnes satisfait $\mathrm{tw}(G) \leq n$.

Remarque. On peut montrer qu'il n'y a pas de décomposition d'une telle grille de largeur inférieure à n. Pour cela on peut utiliser une caractérisation en terme de stratégie entre un voleur et des policiers héliportées (chercher en ligne *cop and robber game*).

Jeu et largeur arborescente.

On considère un jeu sur un graphe G, paramétré par un entier k. Ce jeu à 2 joueurs oppose $k+1$ gendarmes (joueur I) à un voleur (joueur II), vus comme des pions que les joueurs vont disposer sur les sommets du graphe. Intuitivement, les gendarmes héliportés et peuvent soit voler de sommet en sommet, soit rester sur un sommet et construire un barrage ; tandis que le voleur est à moto et doit pouvoir s'échapper indéfiniment en passant d'un sommet à l'autre en traversant les arêtes du graphe sans passer par un barrage.

Plus précisément.

- Au début, le joueur I dispose tous ses gendarmes sur les sommets du graphe avec au plus 1 gendarme par sommet ; et, le joueur II dispose le voleur sur un noeud du graphe où il n'y a pas de gendarme sinon le joueur II perd.
- Ensuite tant que le joueur II n'a pas perdu (tour $i+1$), le joueur I sélectionne certains gendarmes qu'il dispose ailleurs sur le graphe ; et, le joueur I déplace ou non le voleur sur un sommet accessible dans le graphe depuis sa position précédente sans traverser un sommet sur lequel est positionné un gendarme n'ayant pas bougé, sinon il perd.

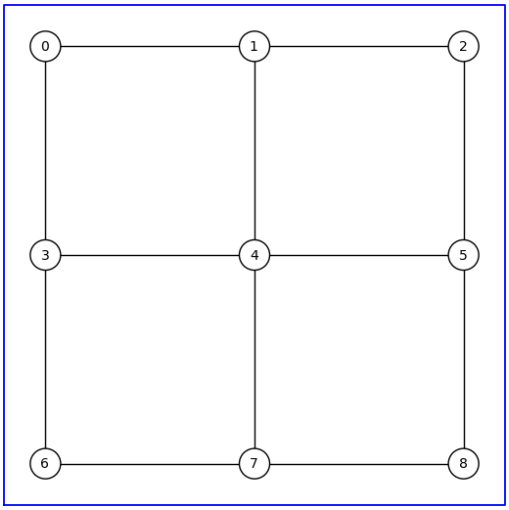
Avec un peu de notation.

- **Tour 0** : I choisit $C_0 \subseteq V(G)$ avec $|C_0| = k+1$; et, II choisit $r_0 \in V(G) \setminus C_0$ ou perd.
- **Tour** $i+1$: I choisit $C_{i+1} \subseteq V(G)$ avec $C_{i+1} = k+1$; et, II choisit $r_{i+1} \in V(G) \setminus C_{i+1}$ tel qu'il y ait un chemin depuis r_i dont les sommets sont dans $V(G) \setminus (C_i \cap C_{i+1})$, sinon il perd.

Le joueur II gagne si il peut s'échapper indéfiniment.

Lemme 3. *Si le joueur I (les gendarmes) a une stratégie gagnante, alors il existe une décomposition arborescente du graphe G de largeur k.*

L'intuition de la preuve : celle-ci se fait par récurrence sur le nombre de tours de jeux maximum avant qu'on attrape le voleur. La condition de début du jeu fait que c'est clair lorsque il y a 0 tours / une décomposition a un seul sac C_0. Pour la partie récurrente, on applique le résultat à $G \setminus C_0$. Selon les choix du joueur II pour r_0, on obtient par récurrence une décomposition d'une des composantes connexes de $G \setminus C_0$: il suffit de recoller ces décompositions autour du sac C_0.



Question 1. Montrez l'implication inverse, à savoir si il existe une décomposition arborescente du graphe G de largeur k alors le joueur I (les gendarmes) a une stratégie gagnante.

Question 2. On considère le graphe d'une grille 3×3 (voir figure ci-contre). Montrez que le joueur II (le voleur) a un stratégie gagnante dans le jeu qui l'oppose à $k+1 = 3$ gendarmes.

Question 3. Expliquer sans rentrer dans les détails comment calculer qui des policiers ou du voleur a une stratégie gagnante (vous pouvez vous appuyez sur un cas concret, en esquissant par exemple une partie du calcul pour la grille 3×3).

Décompositions gentilles

Il sera pratique pour le design d'algorithmes / preuve de leur fonctionnement correct de travailler sur ces décompositions particulières.

Il s'agit de décompositions qui sont enracinées ayant 4 sortes de sacs :

- Les sacs feuilles contenant un seul élément ;
- Les sacs oublis ont pour unique descendant un sac ayant exactement un élément de moins ;
- Les sacs ajouts ont pour unique descendant un sac ayant exactement un élément de plus ; et,
- Les sacs de jointure ayant exactement 2 descendants, tous deux ayant les mêmes éléments.

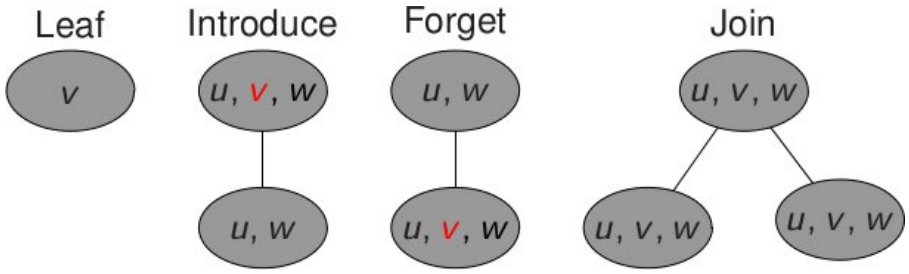


FIGURE 6 – Les 4 types de noeuds d'un arbre d'une décomposition gentille.

Exercice 8. Montrer qu'on peut construire à partir d'une décomposition $\mathcal{T}$ de largeur k d'un graphe G à n sommets une gentille décomposition $\mathcal{T}'$ de largeur k ayant au plus $O(k.n)$ sommets (sacs).

Extension de notre exemple

Reprenons l'exemple du *party problem*, formellement il s'agit de trouver un ensemble indépendant de poids maximal d'un graphe. Cette fois, on dépasse l'organisation hiérarchique purement

arborescente, et on considère des organisations plus réalistes : on suppose que cet organigramme peut être représenté par un graphe ayant une décomposition arborescente de petite largeur k.

Exercice 9. On considère comme instance le graphe de la figure 5, dont les poids seront égaux au numéro du sommet plus votre âge modulo 3.

- Calculez un ensemble indépendant de poids maximal pour cette instance.
- Recommencez en vous appuyant sur une décomposition. Remarque : on peut essayer de rendre les décompositions plus simples à manipuler quitte à avoir un arbre de décomposition un peu plus grand, voir la section précédente sur les décompositions gentilles (vous n'êtes pas obligés d'aller jusqu'à l'étape qui ajoute des feuilles)

6 Algorithmes utilisant la décomposition

Un autre exemple : la colorabilité

ℓ-col For Tree decompositions
- **paramètre** : k
- **instance** : G et une décomposition arborescente de G de largeur au plus k.
- **question** : Est-ce-que G est ℓ-colorable ?

Exercice 10. Montrez que ce problème est polynomial en la taille du graphe d'entrée n et exponentiel en k.

Indication : inspirez vous du travail sur le party problem et son extension (maximum weighted independent set) ; utilisez une approche *bottom-up* en trouvant la bonne quantité d'information à stocker/calculer sur chaque noeud de l'arbre.

Et l'énumération dans tout ça ?

Exercice 11. Reprendre les problèmes de la question précédentes mais cette fois en considérant la version énumérative du problème. C'est-à-dire pour des prédicats R de $I \times O$ où I consiste en l'ensemble des paires un graphe et sa décomposition de largeur bornée par k et O consiste en l'ensemble des solutions correspondants au problème étudié (ℓ-coloriage ou vertex cover minimal).

Quel délai pouvez vous garantir entre 2 solutions ?

7 Pour quels problème peut-on toujours utiliser la décomposition ?

Tout NP ?

On peut se poser la question naturelle de la limite de cette approche. En particulier, *peut-on prendre n'importe quel problème de NP et garantir que ce problème devient polynomial si on a une décomposition arborescente de largeur bornée ?*

Non. Ce n'est pas vrai par exemple pour Edge-disjoint paths (NP-complet pour une largeur de 2) ou encore pour Weighted ℓ-coloring (NP-complet pour une largeur de 3).

Ces problèmes sont définis ci-dessous.

Edge Disjoint Paths With Tree Decomposition
- **paramètre** : k
- **instance** : un graphe G, une décomposition de largeur au plus k, et ℓ paires de sommets distincts s_i, t_i.
- **question** : peut-on trouver des chemins de s_i à t_i qui soient disjoints 2-à-2 pour les arêtes ?

Weighted ℓ-coloring With Tree Decomposition
- **paramètre** : k
- **instance** : un graphe G, une décomposition de largeur au plus k, et une fonction de poids sur les arêtes de G $w : E(G) \to \{1, 2, \ldots, \ell\}$
- **question** : peut-on trouver une fonction $c : V(G) \to \{1, 2, \ldots, \ell\}$ telle que pour toute arête $e = \{x, y\}$ de $E(G)$ on a $|c(u) - c(v)| \geq w(e)$?

Méthode OK pour MSO

On voudrait donc comprendre *pour quels problème de NP notre méthode* bottom-up *s'applique ?* Évidemment, on pourrait lister de nombreux problèmes et pour chacun d'eux chercher à adapter notre approche. Il existe toutefois un résultat très général qui permet de montrer que beaucoup de problèmes deviennent polynomiaux lorsque on les donne via une décomposition arborescente. Ce résultat fait intervenir une logique qu'on appelle *la logique Monadique du second ordre* (MSO) pour faire court. Dans l'immédiat, vous pouvez imaginer MSO comme une sorte de langage de programmation plutôt riche bien qu'il ne permet pas de programmer absolument ce qu'on veut et dont les programmes peuvent tous être exécutés en espace polynomial (dans la classe de complexité Pspace). Ce langage permet d'exprimer des propriétés sur les graphes qui parlent d'ensembles de sommets ou d'arêtes.

Théorème 4 (Courcelle)**.** *Soit k une constante fixée. Tout problème MSO est polynomial lorsque l'entrée G est accompagnée d'une décomposition de largeur au plus k.*

C'est quoi MSO ? Ce langage généralise *la logique du premier ordre*, c'est-à-dire les formules qu'on peut écrire à l'aide des quantificateurs existentiels ($\exists$) et universels ($\forall$) qui porteront sur des variables qui sont des sommets du graphe d'entrée, des connecteurs logiques usuels conjonction ($\wedge$) disjonction ($\vee$) et négation ($\neg$) et donc tous les connecteurs qu'on peut simuler ($\implies$, $\iff$, xor ...), et des tests sur les arêtes du graphe (on écrira $E(x, y)$ pour dire qu'il y a une arête entre x et y).

On peut voir la logique du premier ordre comme un langage de programmation permettant d'écrire des tests vérifiant des conditions locales sur le graphe d'entrée pour certains sommets ou arêtes. Par exemple, "il existe un triangle dans mon graphe" ou encore "il existe un sommet

x qui est voisin de tous les autres sommets" ou encore "il existe 42 sommets tels que toute arête du graphe est incidente à un de ces 42 sommets" Par contre, on ne peut pas exprimer des choses plus globales comme "mon graphe est connexe" mais on peut dire des choses bornées comme "mon graphe a un diamètre de 321 ou moins" en écrivant quelque chose comme "Pour toute paire de sommets x et y, il existe 320 autres sommets qui forment un chemin de x à y ou il existe 319 sommets qui forment un chemin de x à y ou ... ou x et y sont voisins ou ils sont égaux".

Pour pallier à cette faiblesse, on peut ajouter des quantificateurs qui portent sur des ensembles de sommets ou des ensembles d'arêtes[5]. Dans le jargon, on ne dit pas "quantificateur sur un ensemble" mais "quantificateur sur un prédicat monadique". On obtient ainsi la fameuse *logique monadique du second ordre*[6] qu'on notera par MSO. On peut alors exprimer des choses plus globales comme "mon graphe n'est pas un arbre" puisqu'on peut exprimer que "mon graphe contient un cycle" de la manière suivante.

```
Il existe un sous-ensemble de sommets C                 // MONADIQUE EXISTENTIEL
tel que
Il existe au moins 1 élément dans C                    // SUITE DU PREMIER ORDRE
et
Tout sommet v de C a au moins 2 voisins distincts eux aussi dans C.
```

Comme vous pouvez le voir avec l'exemple ci-dessus, ce n'est pas forcément un langage très facile à manipuler sans entraînement. Par contre, comme dans un langage de programmation classique, on peut avoir une approche modulaire et écrire informellement

```
Il existe un ensemble d'arêtes C formant un arbre
```

comme abréviation pour la formule précédente.

Comme autre exemple, on peut exprimer qu'un graphe n'est pas connexe (chose qu'on ne peut pas exprimer avec la logique du premier ordre).

```
Il existe 2 sommets distincts x,y
Il existe un sous ensemble de sommets Cx                // MONADIQUE EXISTENTIEL
tel que
Cx contient x
Cx ne contient pas y
Pour tout sommets adjacents y et z
y appartient à Cx
ssi
z appartient à Cy
```

On peut donc, même sans expérience, se munir d'une boîte à outil MSO permettant de mieux cerner ce qu'on peut exprimer en MSO, de même qu'un programmeur n'a pas nécessairement besoin de savoir implanter les fonctions d'une librairie qu'il appelle.

On peut très facilement exprimer des coloriages, des partitions, en particulier sur les sommets.

[5]Dans la littérature, on fait une distinction fine entre le cas où on ne peut pas quantifier sur les ensembles d'arêtes qui est noté MSO1 et le cas plus général qu'on considère ici qui est noté par MSO2.

[6]Ici le *second* ordre veut dire qu'on a des quantificateurs sur des choses qui ne sont pas des éléments du graphe, mais plus généralement des relations entre ces éléments.

Exercice 12. Décrire informellement un programme MSO pour exprimer qu'un graphe est 3 colorable.

Exercice 13. Adaptez le "programme" MSO ci-dessus pour exprimer que H (un ensemble d'arêtes) forme un *arbre couvrant* du graphe en entrée.

Remarque. Une limite de MSO est qu'on ne peut pas compter. En fait on ne peut même pas compter modulo 2 : il n'existe pas de formule MSO permettant d'exprimer que le graphe a un nombre pair de sommets.

Un résultat plus général.

En fait le théorème de Courcelle est beaucoup plus général que le résultat ci-dessus. Il s'agit d'une version uniforme du résultat ci-dessus. Il existe un algorithme qui prend en entrée, à la fois la formule MSO φ (comprendre le programme MSO), le graphe sur lequel on veut l'appliquer G et une décomposition et donne en sortie la réponse à la question "est-ce-que G satisfait le problème exprimé par φ ?". Cet algorithme qu'on peut intuitivement voir comme une sorte de compilateur MSO / run-time sur graphes de largeur bornée fonctionne en temps $O(f|\varphi| + 2^{p(tw(G))}.|G|)$, où f est une fonction sur les entiers qui est calculable, p est un polynôme et $|\varphi|$ et $|G|$ représente la taille de φ et G respectivement.

Happy end ? Si on examine plus en détails ce résultat il y a des choses très positives lorsque le problème MSO est fixé (ce qui sera notre cas) : on obtient un algorithme qui est FPT (*fixed parameter tractable*) c'est-à-dire où le paramètre est indépendant de la taille n de l'entrée. Intuitivement : la difficulté n'est pas vraiment dans le parcours de l'arbre de décomposition mais isolé dans le calcul associé à chaque sac et pour recoller les morceaux de calculs des fils vers leur père. Cela veut dire que pour un graphe énorme, tant que les sacs ne sont pas trop gros (= ce que vous pouvez vous permettre avec votre ressource de calcul), votre algorithme est applicable.

Non pas vraiment. Il y a par contre une difficulté cachée dans la constante additive $f(k)$, où f est une fonction calculable. Cette fonction croît très vite et donnera des *constantes impraticables* pour des programmes complexes. La preuve de Courcelle repose sur la construction d'automates d'arbres et en particulier l'alternance des quantificateurs donneront lieu à des tours d'exponentielles.

Synthèse. Si le nombre d'alternances de quantificateurs est restreint, on peut tout de même obtenir un prototype pour un problème avec des outils de logique / combinatoire. Ceci reste sans grand intérêt pratique mais permet immédiatement de savoir que l'approche via la décomposition arborescente est théoriquement possible. Ensuite survient un travail pratique d'élaboration d'un algo concret pour le problème qu'on étudie.

8 Calculer une décomposition arborescente

Ce problème est NP-complet.

TREE-WIDTH
- **instance** : un graphe G, un entier k
- **question** : est-ce-que $\text{tw}(G) \leq k$?

Par contre celui-ci est polynomial.

PARAMETERISED TREE-WIDTH
- **paramètre** : k
- **instance** : un graphe G
- **question** : est-ce-que $\text{tw}(G) \leq k$?

En fait il existe un algorithme dû à Bodlaender qui étant donné un graphe G calcule une décomposition de largeur optimal (égale à $\text{tw}(G)$) en temps $O(2^{p(\text{tw}(G))}.|G|)$, où p est un polynôme de degré 3.

Ceci montre que le problème est FPT dans sa version paramétrée. En pratique, toutefois cette dépendance en $2^{(k^3)}$ limite vraiment la grandeur du paramètre k pour lesquels on peut se permettre de calculer exactement une décomposition.

Une approche alternative consiste à utiliser un autre algorithme basé sur les séparateurs qui lui est en $O(2^{3.\text{tw}(G)}.\text{tw}(G).|G|^2)$. Ce second algorithme ne calcule pas une décomposition de largeur k mais une approximation (décomposition de largeur $3k+2$). Finalement, il existe des heuristiques permettant de décomposer les graphes, mais sans garantie d'exécution en temps polynomial.

Conséquence Tous les problèmes de ce cours où j'ai écrit pour l'instance un graphe et sa décomposition et pour paramètre sa largeur restent FPT *si on ne donne pas la décomposition.* On calcule par exemple une approximation de sa décomposition en temps $O(2^{3.\text{tw}(G)}.\text{tw}(G).|G|^2)$, c'est-à-dire une décomposition de largeur au plus $3k+2$ puis on applique la méthode bottom-up associée au problème qui fonctionne en temps exponentiel en la taille des sacs pour chaque sac (ici $3k+2$) lors du traitement d'un sac, multiplié par la taille de l'arbre de décomposition (qui est linéaire en la taille du graphe). Par exemple, pour la 3 colorabilité qui est naïvement en $3^{\text{nbre de sommets}}$, on obtient un temps total de la forme $O(2^{3.\text{tw}(G)}.\text{tw}(G).|G|^2 + 3^{(3k+2)}.|G|)$.

Dans une perspective pratique, il faudra étudier expérimentalement différentes implantations selon les instances typiques qu'on veut résoudre : il sera alors plus ou moins intéressant de travailler sur des décompositions de largeur très proche de $\text{tw}(G)$ ou non.

9 Pour aller plus loin

Je vous invite à découvrir la logique monadique dans les premiers chapitres du livre de Courcelle et Engelfriet [1] (je peux vous aider à trouver une version de prépublication gratuite en ligne). Le livre de Flum et grohe [2] sur la complexité paramétrée vous en apprendra plus sur le sujet, en particulier le chapitre 11 sur les décompositions arborescentes, et comment on les calcule (je peux vous aider à trouver une version gratuite de ce chapitre 11).

Références

[1] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[2] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, March 2006.

Sources

De nombreuses figures de ce script ont été glané sur divers cours disponible sur internet dans des documents mis à disposition par Daniel Marx, Martin Grohe et Michael Elberfeld.