Résumé Partie I du Cours du M2 E-secure
Compter, énumérer, générer

Florent Madelaine

1 Organisation et plan du cours

- 4 séances Florent Madelaine (avant noël)
  - Introduction. Polynôme chromatique
  - Décomposition arborescentes et applications (3 cours)
- 4 ou 5 séances Julien Clément (après noël)
  - Série génératrice (2 cours)
  - Génération aléatoire par méthode récursive
  - comptage probabiliste (1 ou 2 cours)

Évaluation : devoir à la maison + examen final.

2 Motivation

Vous avez par le passé rencontré des problèmes combinatoires simples à énoncer mais théoriquement difficiles à résoudre sur de grandes instances qui capturent l'essence de problèmes plus concrets qu'on rencontre de manière routinière dans l'industrie et qu'on se doit de modéliser et résoudre (au moins sous forme approchée).

Par exemple, le problème de *coloriage d'un graphe* n'est pas très différent du problème d'*allocation de fréquences* qu'une antenne doit résoudre pour permette aux utilisateurs d'un téléphone portable de pouvoir communiquer en évitant les interférences. Le problème de *satisfiabilité d'une formule propositionnelle* (SAT) qui est l'archétype d'un problème difficile[1] semble à première vue plutôt abstrait, mais ce formalisme permet de modéliser de nombreux problèmes concrets et les *solveurs sat* permettent de résoudre à présent de grandes instances industrielles (plusieurs millions de variables).

Le plus souvent, vous avez tâché de *trouver une solution* à une instance du problème, ou encore de savoir si il *existe une telle solution*. Il existe de nombreux contextes dans lesquels on veut pouvoir avoir de nombreuses solutions, voir *toutes les solutions*. Par exemple, en bases de données l'utilisateur veut fréquemment obtenir plusieurs solutions, voir toutes les solutions. Parfois, il lui suffit de connaître le *nombre de solutions* de sa requête. On résout aussi souvent des problèmes d'optimisation, par exemple pour l'allocation de fréquences, il ne suffit pas que les fréquences soient différentes, il faut aussi que la bande passante totale soit minimale (l'opérateur paye la bande passante) et que des utilisateurs proches aient des fréquences éloignées (pour éviter les interférences). Même si on s'intéresse le plus souvent à des approximations du problème d'optimisation initial en utilisant des heuristiques, il faut pouvoir énumérer plusieurs solutions.

Pour pouvoir mieux appréhender l'efficacité de solveurs sat, il est souvent utile de pouvoir savoir *générer* des instances de taille variables ressemblant à des instances industrielles typiques[2].

[1] Il est NP-complet, c'est historiquement le premier problème de ce type par le théorème de Cook.

[2] En anglais, on parle de *benchmark*

Il existe aussi de nombreuses méthodes aléatoires permettant de trouver des solutions approchées lorsque la recherche d'une solution exacte s'avère trop élusive. Il faut donc savoir générer aléatoirement certains objets combinatoires. Finalement, lorsqu'on veut pouvoir dépasser le cadre un peu restrictif de la complexité pire des cas (cadre dans lequel vous cous êtes *a priori* toujours placé) et qu'on veut analyser la complexité en moyenne, il est important de pouvoir générer des structure combinatoires.

Ce cours s'intéresse donc au delà de l'existence d'une solution à une instance d'un problème, à compter les solutions, les énumérer ou encore les générer aléatoirement.

3 Problématique

Notation

Pour un problème combinatoire fixé,

- soit I l'espace des instances du problème ; et,
- soit O l'espace de ces solutions

On considère le prédicat binaire $R \subseteq I \times O$ modélisant la correspondance entre une instance et sa ou ses solutions.

Exemple. Pour le problème de k-colorabilité, I est l'ensemble des graphes non orientés finis et O l'ensemble des fonctions de l'ensemble des sommets vers l'ensemble des couleurs.

Un graphe G de sommets $V(G)$ est en correspondance avec une fonction de coloriage f de $V(G)$ dans $\{1, 2, \ldots, k\}$, ce qu'on note $(G, f) \in R$ ou encore $R(G, f)$ ssi f est une fonction de coloriage *admissible* (sommets voisins dans G ont une couleur distincte).

On note $R(x) = \{y | (x, y) \in R\}$ l'ensemble des solutions du problème R sur l'instance x. On note $\sharp S$ le nombre d'éléments d'un ensemble S. En particulier, $\sharp R(x)$ est le nombre de solutions de R pour l'instance x.

DECIDE(R)
- **Entrée** : $x \in I$
- **Sortie** : accepte ssi $R(x) \neq \emptyset$

COMPTE(R)
- **Entrée** : $x \in I$
- **Sortie** : $\sharp R(x)$

ENUM(R)
- **Entrée** : $x \in I$
- **Sortie** : une énumération $y_1, \ldots, y_n$ de $R(x)$

Exercices

Exercice 1. Lorsque R est la relation R_{3col} correspondant au problème de 3-coloriage, et l'instance x est un chemin de longueur 2, indiquez la sortie pour les trois problèmes DECIDE(R_{3col}), COMPTE(R_{3col}) et ENUM(R_{3col}).

Même question lorsque x est un triangle.

Même question lorsque x est un triangle plus (union disjointe) un chemin de longueur 2.

Exercice 2. On change de problème et on s'intéresse au problème SAT. On note R_{sat} le prédicat correspondant. On considère la formule propositionnelle suivante.

$$\varphi := (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_2 \vee \neg x_4) \\ \wedge (x_2 \vee \neg x_3) \wedge (\neg x_4 \vee \neg x_3) \wedge (x_3) \wedge (x_5 \vee \neg x_1 \vee \neg x_2 \vee \neg x_3)$$

Indiquez la sortie pour les trois problèmes DECIDE(R_{sat}), COMPTE(R_{sat}) et ENUM(R_{sat}) pour cette entrée.

Exercice 3. Pour la relation R d'un problème fixé, trois étudiants doivent programmer les trois problèmes. Ces trois étudiants D,E et C programme respectivement en premier DECIDE(R), COMPTE(R) et ENUM(R). En supposant qu'ils ne partagent pas leur travail entres eux mais qu'ils réutilisent si possible leur propre travail, expliquez comment chacun d'entre eux procède.

Exercice 4. Pour la relation R_{Sat}, un étudiant hérite du code pour DECIDE(R_{sat}). Comment peut-il procéder pour réutiliser ce code pour programmer COMPTE(R) et ENUM(R) ?

4 Polynôme chromatique

Vous verrez plus tard avec Julien Clément que l'analyse permet d'appréhender le dénombrement et la génération aléatoire de structures combinatoires, en passant par les séries entières.

Nous allons voir un résultat d'un esprit similaire mais beaucoup plus simple. Pour un graphe G, on peut toujours calculer un *polynôme* de variable entière k qu'on notera $p_G(k)$ dont la valeur en k est précisément le nombre de k-coloriages admissibles (soit $P_G(k) = \sharp R_{kCol}(G)$ avec la notation introduite précédemment.).

La preuve qu'il s'agit bien d'un polynôme est effective : c'est-à-dire qu'elle va nous donner une méthode pour le calculer.

Existence et méthode de calcul du polynôme chromatique

Les ingrédients nécessaires à la preuve d'existence et au calcul de ce polynôme sont les suivants.

L'idée principale de l'algorithme de calcul du polynôme chromatique consiste à observer que étant donné un graphe G et deux sommets x et y non-adjacents, on a deux types de coloriages : ceux pour lesquels la couleur de x et la même que celle de y, et ceux pour lesquels la couleur ce x et de y sont différentes. Autrement dit le premier type correspond à des coloriages du graphe $G_{x=y}$ (le graphe obtenu en identifiant x et y en un seul sommet) et le second type correspond à des coloriages du graphe $G_{+\{xy\}}$ (le graphe obtenu en ajoutant l'arête $\{x, y\}$). Le nombre total de coloriages est la somme de ces deux cas. On a donc le résultat suivant.

Lemme 1. *Soit G un graphe et soient x et y deux sommets* qui ne sont pas adjacents.
Si on connaît les polynômes chromatiques de $G_{x=y}$ et de $G_{+\{x,y\}}$, alors on connaît celui de G. En effet,

$$P_G(k) = P_{G_{x=y}}(k) + P_{G_{+\{x,y\}}}$$

La seconde idée et d'observer qu'on peut colorier indépendemment les composantes connexes d'un graphe.

Lemme 2. *Si on connaît les polynômes chromatiques de H et G, alors on connaît celui de $H + G$. En effet,*

$$P_{H+G}(k) = P_H(k)P_G(k)$$

Finalement on a besoin de connaître le cas des cliques.

Lemme 3. *Pour un graphe complet K_n, on a $P_{K_n}(k) = k(k-1)\cdots(k-n+1)$.*

Ces trois ingrédients permettent de prouver correct l'algorithme ci-dessous.

Algorithme pour le calcul du polynôme chromatique

1. `Si le graphe n'est pas connexe alors prendre le produit des polynômes chromatiques de ces composantes connexes`
2. `Si le graphe est connexe mais pas complet alors choisir deux sommets` x `et` y `et faire la somme des polynômes chromatiques de` $G_{+\{xy\}}$ `et` $G_{x=y}$.
3. `Si le graphe est complet alors prendre` $P_{K_n}(k) = k(k-1)\cdots(k-n+1)$.

Théorème 4. *Existence et calcul du polynôme chromatique Pour tout graphe G, le polynôme chromatique $P_G(k)$ existe et est calculé par l'algorithme ci-dessus.*

Remarque. Il existe d'autres polynômes pour les graphes, dont le célèbre *polynôme de Tutte* à partir duquel on peut retrouver le polynôme chromatique.

Exercices

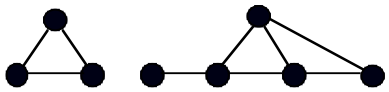
Exercice 5.

Montrer que $P_{K_n}(k) = k(k-1)\cdots(k-n+1)$

Montrer que $P_{I_n}(k) = k^n$, où I_n est le graphe consisté de n sommets isolés.

Exercice 6. Calculer le polynôme chromatique du graphe G_0 obtenu en ajoutant un sommet adjacent à un sommet de K_3.

Exercice 7. Déterminer le polynôme chromatique du graphe G ci-dessous.



Exercice 8. Donner une définition du nombre chromatique à partir de celle du polynôme chromatique.

Petite remarque historique. Introduit seulement pour des graphes planaires en 1912 par George David Birkhoff, un mathématicien américain qui cherchait à démontrer analytiquement le théorème des 4 couleurs (ce théorème, démontré beaucoup plus tard à l'aide d'un ordinateur, stipule que tout graphe planaire peut être colorié avec 4 couleurs de sorte que 2 sommets adjacents soient de couleur différente).

Exercice 9. Proposer une méthode alternative pour calculer le polynôme chromatique lorsque le graphe G est dense (a plutôt beaucoup d'arêtes). Indication : inverser la règle de calcul du Lemme 1.

Exercice 10. Déduire de l'exo précédent une variante de l'algo du polynôme chromatique qui est en temps $O(2^{n+m})$ où n est le nombre de sommets de G et m son nombre d'arêtes.

Remarque. On peut améliorer l'analyse de l'exo précédent et obtenir le nombre d'or à la place de 2 (en $O(\varphi^{n+m})$ où $\varphi = \frac{1+\sqrt{5}}{2} \equiv 1.62$). La technique est similaire à l'analyse de la récurrence naturelle pour calculer la célèbre suite de Fibonacci.

Exercice 11. Montrer que le polynôme chromatique d'un arbre T à n sommets est $P_T(k) = t(t-1)^{n-1}$. Montrer que $P_{C_n}(k) = (t-1)^n + (-1)^n(t-1))$.

5 Décompositions arborescentes

Il s'agit de notions issues de la théorie des graphes, étudiées depuis le début des années 70, qui ont 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*).

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).

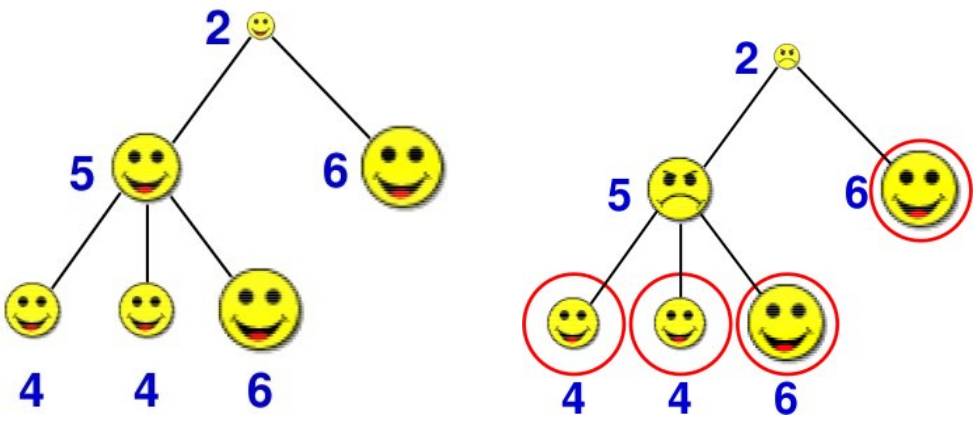


Figure 1 – Instance et Solution du Party Problem

Exercice 12. 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).

Définition informelle

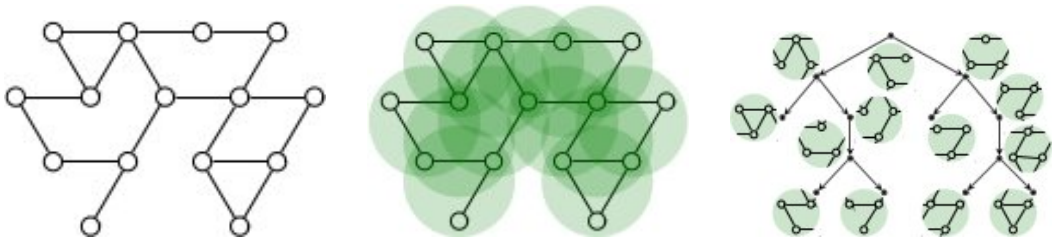


FIGURE 2 – Graphe, recouvrement par des sacs, arbre de sacs.

- 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)

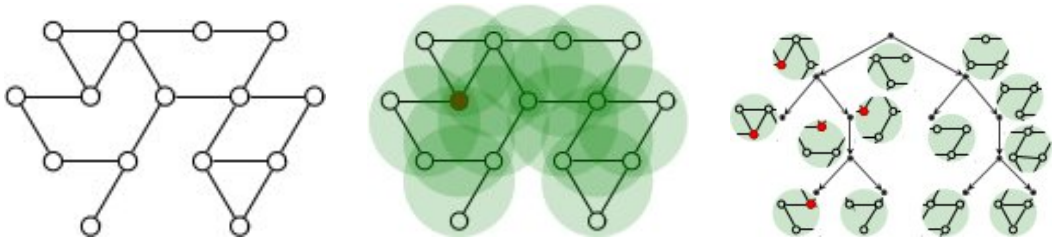


FIGURE 3 – 1 sommet correspond à un sous-arbre.

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*[3].

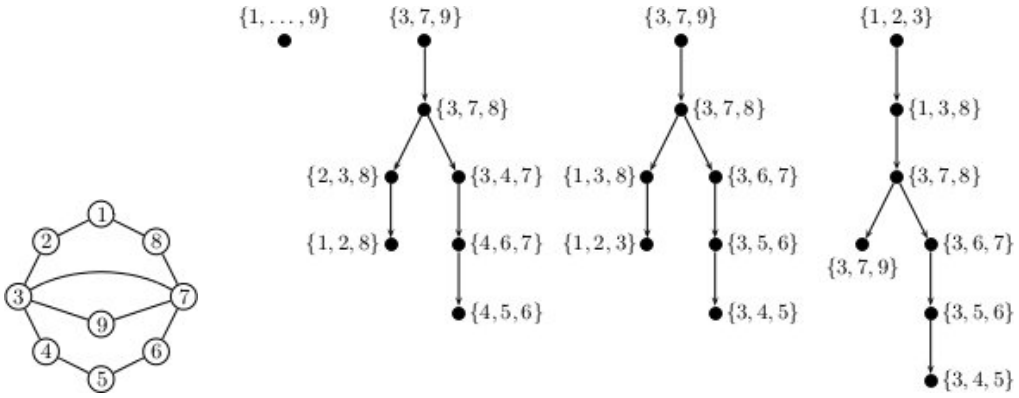


FIGURE 4 – Un graphe peut avoir plusieurs décompositions.

Exercice 13. Montrer que pour un arbre T, on peut toujours trouver une décomposition de largeur 1.

Exercice 14. 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 15. Quel est la largeur d'arborescence d'un arbre ? d'un cycle à n sommets, $n \geq 3$? d'une clique K_n, $n \geq 2$?

Lemme 5. *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 16. 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.

Exercice 17. 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.

Exercice 18. Montrer que un graphe de grille de n lignes et n colonnes satisfait $\mathrm{tw}(G) \leq n$.

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

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*).

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.

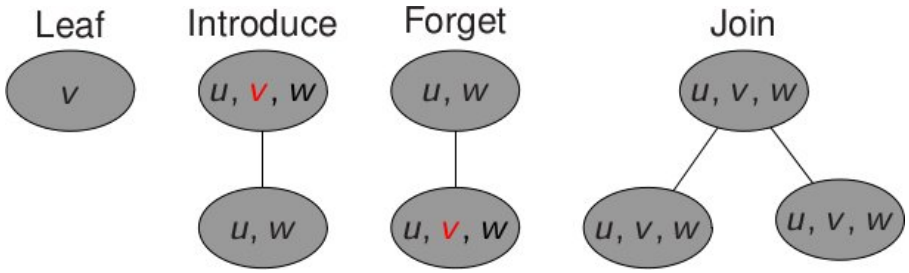


FIGURE 5 – Les 4 types de noeuds d'un arbre d'une décomposition gentille.

Exercice 19. 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 20. On considère comme instance le graphe de la figure 4, 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 21. 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.

Un autre exemple : *vertex cover*

Vertex Cover For Tree decompositions
- **paramètre** : k
- **instance** : G et une décomposition arborescente de G de largeur au plus k.
- **output** : un vertex cover de taille minimale

Exercice 22. Montrez que ce problème est polynomial en la taille du graphe d'entrée n et exponentiel en k.

Et l'énumération dans tout ça ?

Exercice 23. 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 6 (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 ($\Longrightarrow$, $\Longleftrightarrow$, 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[4]. 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*[5] 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
```

[4]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.

[5]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.

```
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.

Exercice 24. Décrire informellement un programme MSO pour exprimer qu'un graphe est 3 colorable.

Exercice 25. 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

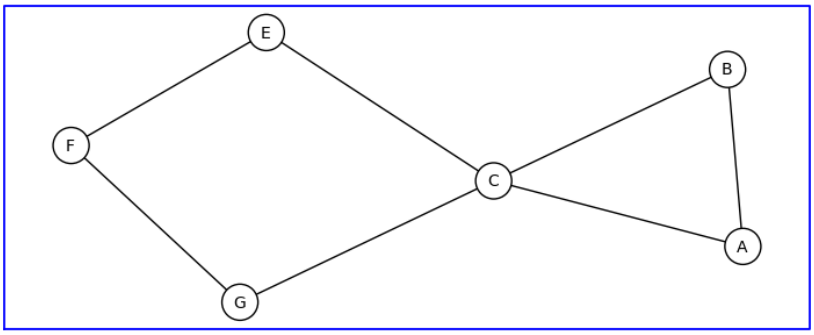
De nombreuses figures de ce script ont été glané sur divers cours disponible sur internet dans des documents mis à disposition par Daniel Marx, Martin Grohe et Michael Elberfeld.

Devoir à la maison

à rendre pour le 9 janvier
une soutenance *individuelle* de 5 à 10 minutes organisée ultérieurement
permettra de juger de la compréhension de ce que chaque étudiant a rendu

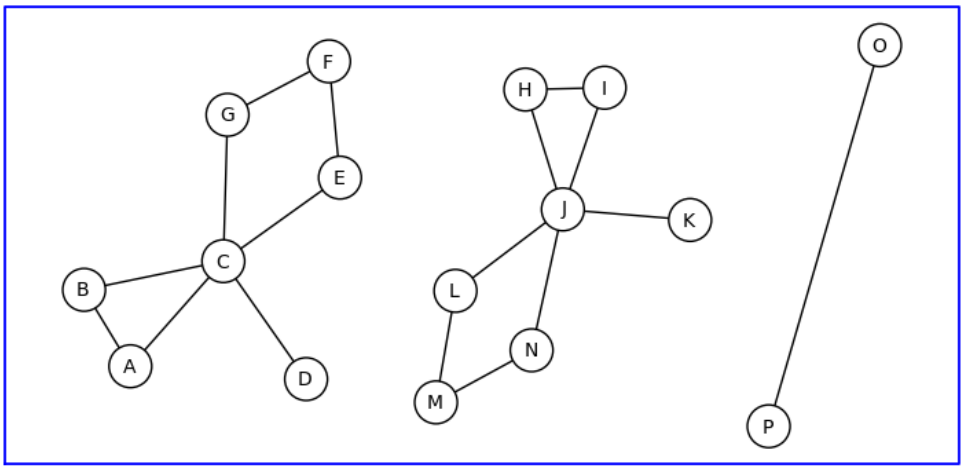
Exo 1 (polynôme chromatique)

On admettra que le graphe ci-dessous a pour polynôme chromatique $(x-2)\cdot x\cdot(x-1)^2\cdot(x^2-3x+3)$



En déduire que le polynôme chromatique du graphe ci-dessous est

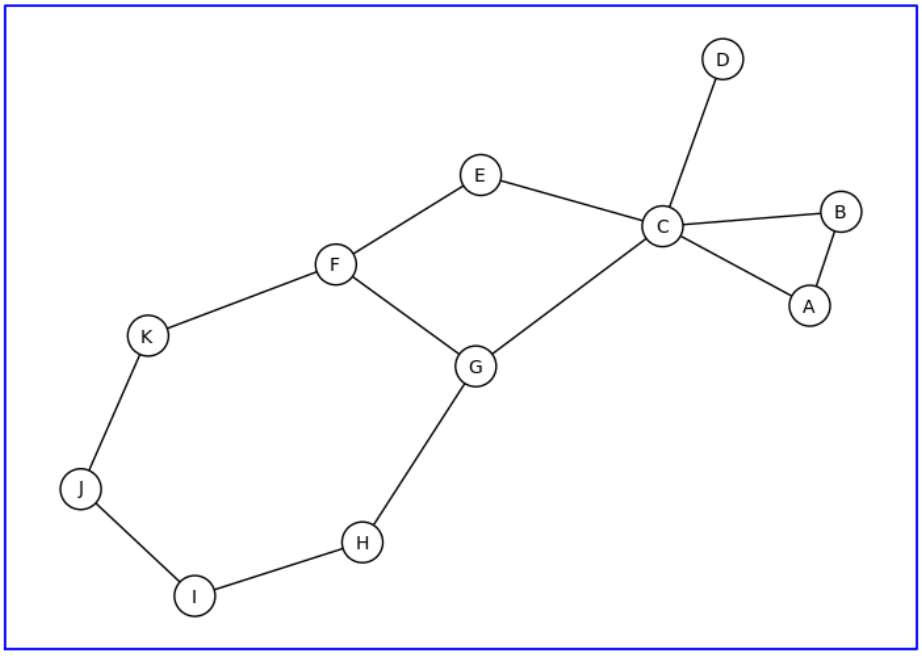
$$(x-2)^2\cdot x^3\cdot(x-1)^7\cdot(x^2-3x+3)^2$$



Indications : on pourra utiliser une méthode proche de celle vue en cours et étudiée en exercice qui relie polynôme chromatique d'un graphe ayant une arête x, y avec celui du graphe obtenu en enlevant cette arête et en la contractant, respectivement.

Exo 2 (décomposition arborescente)

Donnez une décomposition arborescente de largeur minimale pour le graphe ci-dessous, en expliquant bien pourquoi elle est minimale.



Exo 3 (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.

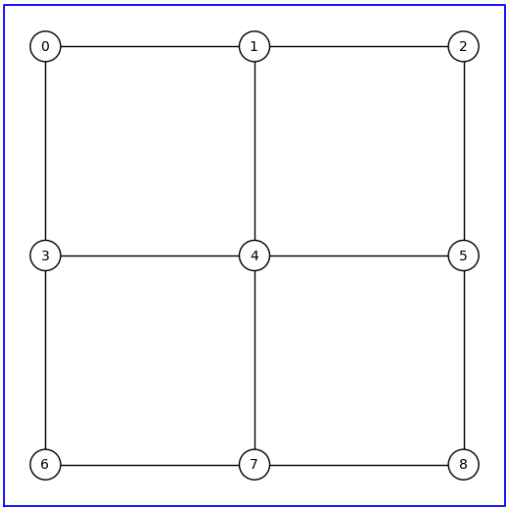
- **Tour 0** : I choisit $C_0 \subseteq V(G)$ avec $|C_0| = k+1$; et, II choisit $r_0 \in V(G) \setminus C_0$ ou perd.
- **Tour** $i+1$: I choisit $C_{i+1} \subseteq V(G)$ avec $C_{i+1} = k+1$; et, II choisit $r_{i+1} \in V(G) \setminus C_{i+1}$ tel qu'il y ait un chemin depuis r_i dont les sommets sont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

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

On admettra le lemme suivant.

Lemme 7. *Si le joueur I (les gendarmes) a une stratégie gagnante, alors il existe une décomposition arborescente du graphe G de largeur k.*

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.

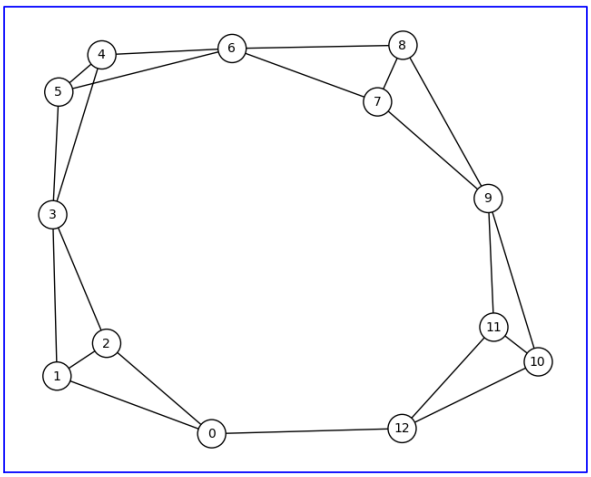
Question 3. Expliquer sans rentrer dans les détails comment calculer qui des policiers ou du voleur a une stratégie gagnante (vous pouvez vous appuyez sur un cas concret, en esquissant par exemple une partie du calcul pour la grille 3×3).

Exo 4 (programmation dynamique)

On considère le problème combinatoire de 3-colorabilité. Programmer la méthode *bottom-up* vue en cours prenant en entrée un graphe G et une décomposition arborescente, permettant selon les désirs de l'utilisateur de décider, générer une solution, compter les solutions ou énumérer les solutions.

Pour tester votre programme, voici 2 instances (graphe + décomposition).

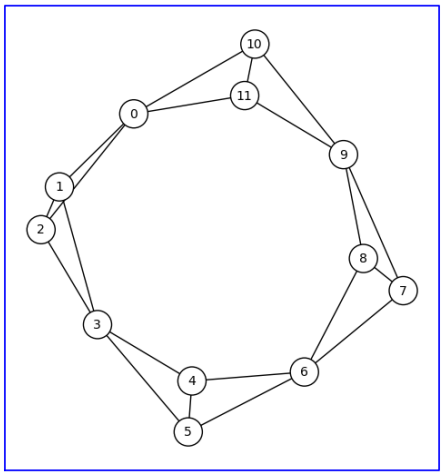
Instance 1.



Avec les sacs :

$A : \{0, 1, 2, 3\}$
$B : \{0, 3, 4, 5\}$
$C : \{0, 4, 5, 6\}$
$D : \{0, 6, 7, 8\}$
$E : \{0, 7, 8, 9\}$
$F : \{0, 9, 10, 11\}$
$G : \{0, 10, 11, 12\}$

Instance 2.



Avec les sacs :

$A : \{0, 1, 2, 3\}$
$B : \{0, 3, 4, 5\}$
$C : \{0, 4, 5, 6\}$
$D : \{0, 6, 7, 8\}$
$E : \{0, 7, 8, 9\}$
$F : \{0, 9, 10, 11\}$