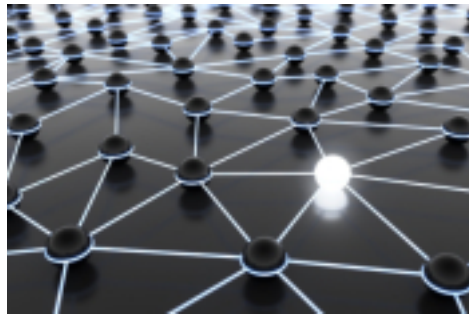


Analyse d'Algorithme



Cyril Gavoille

LaBRI

Laboratoire Bordelais de Recherche
en Informatique, Université de Bordeaux

gavoille@labri.fr

17 mai 2011

Master 2 : Analyse d'Algorithmes

Objectifs : introduction à la complexité paramétrique ; aux algorithmes d'approximation

Pré-requis : algorithmique ; notions de théorie des graphes

Références :

- *Invitation to fixed-Parameter Algorithms*, R. Niedermeier. Springer 2006
- *Parametrized Complexity Theory*, J. Flum, M. Grohe. Springer 2006
- *Parametrized Complexity*, R.G. Downey, M.R. Fellows. Springer 1999
- *Algorithmes d'approximation*, V.V. Vazirani. Springer 2006
- *Approximation Algorithms for NP-hard Problems*, Dorit S. Hochbaum. PWS 1995

Contrôle des connaissances :

- Examen écrit : 1h30
- Note finale : 1/2 Note Examen + 1/2 Projet de Programmation

Table des matières

1	Introduction	5
1.1	À quoi ça sert ?	5
1.2	Quelques algorithmes que nous analyserons	7
1.3	Des algorithmes pour construire quoi ?	7
1.4	Existence d'objets spécifiques: les <i>spanners</i>	8
2	Algorithmes exacts	17
2.1	Temps polynomial <i>vs.</i> exponentiel	17
2.2	Problèmes jouets	18
2.3	Algorithmes exhaustifs (<i>brute force</i>)	19
2.4	Complexité paramétrique: définition	20
2.5	Arbre borné de recherche	23
2.5.1	COUVERTURE PAR SOMMETS de taille k	23
2.5.2	ENSEMBLE INDÉPENDANT pour les graphes planaires	25
2.5.3	ENSEMBLE DOMINANT pour les graphes planaires	27
2.5.4	Améliorations : raccourcissement de l'arbre	32
2.6	Réduction à un noyau : « kernalisation »	35
2.6.1	Principe et propriété	35
2.6.2	Exemple de noyau	36
2.6.3	COUVERTURE PAR SOMMETS	37
2.6.4	Noyau et approximation	41
2.7	Théorie des mineurs de graphes	42
2.7.1	Définitions	42
2.7.2	Théorème des mineurs de graphes	43
2.7.3	Applications	44

2.8	Réduction aux graphes de <i>treewidth</i> bornée	47
2.8.1	Décomposition arborescente	48
2.8.2	Un exemple simple: 3-COLORATION	52
2.8.3	COUVERTURE PAR SOMMETS pour <i>treewidth</i> bornée	54
2.8.4	Application aux graphes planaires	55
2.8.5	Amélioration pour les graphes planaires	58
2.9	Remarques finales	62
3	Algorithmes d'approximation	67
3.1	Introduction	67
3.2	Problèmes bien caractérisés	68
3.2.1	Couplage	68
3.2.2	Les problèmes MIN-MAX	69
3.2.3	Comment concevoir un algorithme d'approximation ?	69
3.3	COUVERTURE PAR SOMMETS de taille minimum	70
3.3.1	Un premier algorithme	70
3.3.2	Un second algorithme	70
3.3.3	Technique de précalcul (<i>color coding</i>)	71
3.4	COUVERTURE PAR ENSEMBLES	73
3.4.1	Algorithme glouton	75
3.4.2	Un second algorithme	78
3.4.3	Algorithme exact	79

1.1 À quoi ça sert ?

Essentiellement à deux choses :

1. À se rendre compte qu'un algorithme ne marche pas très bien (complexité trop grande même sur des données de petite taille), voir pas du tout (algorithme faux ou non conforme). Cela évite ainsi de programmer inutilement.
2. À démontrer l'existence d'objets ou de structures discrètes vérifiant certaines propriétés, en proposant un algorithme de construction et en prouvant sa correction.

Nous illustrerons le deuxième point à la fin de ce chapitre.

Attention! On ne peut pas analyser n'importe quel algorithme. Des algorithmes très simples sont parfois extrêmement difficile à analyser. En voici deux exemples.

Ex1 :

Algorithme GOLDBACH(n)

Entrée : un entier pair $n > 2$

Sortie : un booléen

1. Si $n = 4$, renvoyer VRAI.
2. Pour $i := 1$ à $n/2 - 1$ faire :
 - si $2i + 1$ et $n - (2i + 1)$ sont deux nombres premiers, renvoyer VRAI.
3. Renvoyer FAUX.

Ainsi, pour $n = 30$, l'algorithme va tester la primalité de :

- 3 et 27? NON;
- 5 et 25? NON;
- 7 et 23? OUI.

Cet algorithme teste si le nombre pair $n > 2$ est la somme de deux nombres premiers.

Est-ce que :

$$\forall n > 2 \text{ pair, } \text{GOLDBACH}(n) = \text{VRAI} ?$$

Cette analyse s'annonce *a priori* très difficile car c'est un problème non résolu connu sous le nom de Conjecture de Goldbach. Elle est vérifiée pour tout entier $n < 1.1 \times 10^{18}$ (février 2008).

Ex2 :

Algorithme SYRACUSE(n)

1. Tant que $n > 1$ faire :
 - si n est pair, alors $n := n/2$ sinon $n := 3n + 1$
-

Est-ce que :

$$\forall n \in \mathbb{N}, \quad \text{SYRACUSE}(n) \text{ s'arrête ?}$$

C'est un problème non résolu (problème appelé aussi « $3x + 1$ »). C'est vrai pour tout entier $n < 2^{62}$ (janvier 2008 - T. Oliveira e Silva), ce qui est supérieure à quatre milliards de milliards!

De manière générale, il est indécidable de savoir si un programme ou algorithme donné s'arrête ou boucle indéfiniment. Il ne faut donc pas espérer trouver une technique systématique (un algorithme donc) qui analyse tout algorithme : on peut prouver qu'une telle méthode n'existe pas!

Grégory J. Chaitin a définit le nombre suivant ¹ :

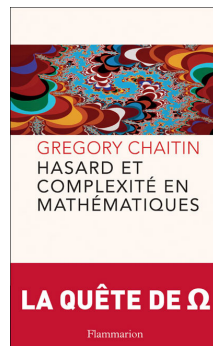
$$\Omega = \sum_{i \geq 1} P_i \cdot 2^{-i}$$

où $P_i \in \{0, 1\}$ vaut 1 si et seulement si le programme numéro i s'arrête (disons les programmes écrits en C classés par ordre alphabétique). On remarque que $0 < \Omega < 1$ et que si l'on écrit Ω en binaire, le i -ème bit après la virgule vaut précisément P_i . En fait, on connaît très peu de chose sur ce nombre. La connaissance de la première centaine de bit d' Ω permettrait de résoudre déjà de très grandes énigmes mathématiques. On pourrait par exemple écrire un programme très compact permettant de vérifier la conjecture de Goldbach :

« $n := 4$; tant que GOLDBACH(n) faire $n := n + 2$ »

Si ce programme boucle, alors la conjecture est vraie. Sinon, un contre-exemple existe et elle est fausse.

1. En fait, il s'agit du nombre de Turing. Les bits de ce nombre contiennent de la redondance, ils sont loin d'être tous indépendants. En effet, $\lceil \log n \rceil$ bits d'information suffisent à décrire les n premiers bits de ce nombre : si exactement p bits des n premiers bits sont à 1 – ce nombre se code en binaire avec au plus $\lceil \log n \rceil$ bits – alors, pour calculer les n premiers bits, il suffit de lancer les n programmes en parallèle et d'attendre que p d'entre eux s'arrêtent. Le nombre Ω de Chaitin est défini de manière similaire à celui de Turing en supprimant la redondance.



1.2 Quelques algorithmes que nous analyserons

- Algorithmes exacts (au chapitre 2)
- Algorithmes d’approximation (au chapitre 3)

Ceux que nous n’analyserons pas :

- Algorithmes probabilistes

1.3 Des algorithmes pour construire quoi ?

Dans notre cours, il s’agira essentiellement « d’algorithmes combinatoires », généralement liés aux graphes. La plupart du temps le problème étudié sera « difficile », c’est-à-dire NP-complet (décision) ou NP-difficile (optimisation).

Ex1 :

HAMILTONISME :

Instance: un graphe G

Question: est-ce que G possède un cycle Hamiltonien ? (un cycle passant une et une seule fois par tous les sommets du graphe)

Un problème d’optimisation associé à HAMILTONISME pourrait-être de trouver dans un graphe la longueur du plus long cycle. Ce problème est NP-complet, c’est-à-dire qu’il appartient à la classe des problèmes NP et que tout problème dans NP se réduit polynomialement à lui.

NP = « Non determinist Polynomial »

Un problème \in NP si pour chacune des instances du problème il existe un certificat positif que l’on peut tester en temps polynomial en la taille de l’instance.

Par exemple, un certificat positif pour HAMILTONISME peut-être un cycle de G . En quelques sortes, si l'on devine (N) la solution, on peut la vérifier en temps polynomial (P). Nous reparlerons des certificats au chapitre 3 concernant les algorithmes d'approximation.

Ex2 :

PRIMALITÉ :

Instance: un entier n

Question: est-ce que n est premier ?

Les algorithmes basés sur un crible d'Eratosthène (240 avant J.-C.), consistant à tester (récursivement) tous les nombres premiers $\leq \sqrt{n}$, ont une complexité au moins $\Omega(\sqrt{n})$. Il faut réaliser que ces algorithmes sont très loin d'être polynomiaux puisque la taille de l'entrée du problème est ici de seulement $\lceil \log n \rceil$ bits². Le crible d'Eratosthène appliqué à un nombre de taille k (en bits) s'exécute en au moins $\sqrt{2^k} = 2^{k/2}$ étapes ...

Contrairement à ce que l'on pourrait penser, ce problème est bien dans P, et donc peut être résolu en temps polylogarithmique en n , cf. [AKS04]. La complexité de leur algorithme est $(\log^{7.5} n)(\log \log n)^{O(1)}$. Le polynôme en $\log \log n$ est lié au fait que les multiplications/divisions sur des nombres de $k = O(\log n)$ bits sont nécessaires. De telles opérations peuvent s'effectuer en temps un peu inférieur à $^3 k \log^2 k$.

Comme bien souvent, l'algorithme est bien plus simple que son analyse. Le voici :

En fait, la complexité réelle de l'algorithme est probablement beaucoup moins. Si la conjecture de Sophie Germain est vraie⁴, la complexité est seulement de $(\log^6 n)(\log \log n)^{O(1)}$. Il y a donc une différence entre le comportement réel de l'algorithme, et la complexité prouvable. En fait, un autre algorithme à été proposé avec la même complexité mais dont l'analyse ne repose sur aucune conjecture.

1.4 Existence d'objets spécifiques : les *spanners*

Pour terminer ce chapitre, nous allons illustrer la construction d'objets non triviaux à l'aide d'algorithmes simples et de leur analyse.

Ces objets sont les *spanners*, que l'on peut traduire par « sous-graphes couvrants » ou « graphes de recouvrement ». L'idée est de construire un sous-graphe couvrant tous les sommets mais ayant beaucoup moins d'arêtes toute en préservant les distances à un facteur α près. On peut voir un *spanner* comme une sorte de squelette d'un graphe.

2. La fonction \log représente le logarithme en base deux.

3. Plus précisément, le meilleur algorithme de multiplication à une complexité de $(k \log k) \cdot 2^{O(\log^* k)}$ où la fonction $\log^* k = \min\{i : \log^{(i)} n \leq 1\}$, cf. [Für09].

4. Cette conjecture affirme qu'il existe une infinité de nombres premiers p tels que $2p + 1$ soit aussi premier, comme 2, 3, 5, 11, 23, 29, 41, 53, 83, 89, 113, 173, 179, 191, 233 ... Le plus grand qu'on ait trouvé (en 2010) a environ 80 000 chiffres.

Input: integer $n > 1$.

1. If $(n = a^b$ for $a \in \mathcal{N}$ and $b > 1)$, output **COMPOSITE**.
2. Find the smallest r such that $o_r(n) > \log^2 n$.
3. If $1 < (a, n) < n$ for some $a \leq r$, output **COMPOSITE**.
4. If $n \leq r$, output **PRIME**.¹
5. For $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ do
 - if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$, output **COMPOSITE**;
6. Output **PRIME**.

FIGURE 1.1 – Test de primalité original selon [AKS04]. Ils montrent qu'à l'étape 2, $r \leq \lceil \log n \rceil^5$ si bien que le test de l'étape 4 n'a d'intérêt que si $n \leq 5\,690\,034$ (c'est l'objet de la note de bas de page ¹ de l'article en ligne 4). L'entier $o_r(n)$ représente le plus petit entier k tel que $a^k = 1 \pmod{r}$. La notation (a, n) de l'étape 3 représente le plus petit commun diviseur de a et n . Enfin, $\phi(r)$ représente la fonction d'Euler, c'est-à-dire le nombre d'entiers $< r$ premiers avec r . Il faut noter que $\sqrt{\phi(r)} \leq \sqrt{r} \leq \lceil \log n \rceil^{5/2}$.

Définition 1 *Un sous-graphe H d'un graphe G est un α -spanner si pour toute paire de sommets $x, y \in V(G)$, $d_H(x, y) \leq \alpha \cdot d_G(x, y)$.*

Ici, $d_H(x, y)$ représente la distance dans H entre x et y . Si les arêtes du graphe sont valuées si s'agira du coût minimum (somme des coûts) d'un chemin reliant x à y dans H (si aucune précision n'est donnée, les arêtes ne sont pas valuées).

Le paramètre α est appelé *étirement*, et le nombre d'arêtes de H , sa *taille*. L'objectif dans l'étude des *spanners* est généralement de minimiser la taille pour un étirement α donné.

En prenant $H = G$, on a évidemment que tout graphe possède un *1-spanner* dont la taille est identique à celle de G , donc $O(n^2)$. D'un autre côté, tout graphe connexe à n sommets possède un *spanner* de taille $n - 1$, un arbre couvrant. Cependant, l'étirement peut être aussi grand que $n - 1$, pour un cycle par exemple.

On va montrer le résultat suivant :

Théorème 1 *Tout graphe à n sommet possède un 3-spanner de taille au plus $n^{3/2} - n/2$.*

Avant de donner une preuve (constructive) de ce résultat, on introduit la définition suivante qui nous servira dans le reste du cours.

Notations : On note $B_G(u, r)$ la boule (ou sous-graphe induit) de rayon r et centrée en u dans le graphe G .

Preuve. La notation $\text{BFS}_G(u, H)$ (*Breadth First Search*) représente un arbre en largeur d'abord dans G de racine u et couvrant les sommets de H . S'il n'est pas possible de couvrir entièrement H (par exemple si H est réparti sur plusieurs composantes connexes de G), alors seuls les sommets de H appartenant à la composante connexe de G contenant u sont couverts. Lorsque G est sous-entendu, on note plus simplement $\text{BFS}(u, H)$.

On considère l'algorithme suivant :

Algorithme 3SPANNER(G)

Entrée : un graphe G

Sortie : un *spanner* H

1. $H := (\emptyset, \emptyset)$, le graphe vide
 2. Tant qu'il existe $u \in V(G)$, $\deg_G(u) \geq \sqrt{n}$ faire :
 - (a) $H := H \cup \text{BFS}_G(u, B_G(u, 2))$
 - (b) $G := G \setminus B_G(u, 1)$
 3. $H := H \cup G$
-

On va d'abord montré qu'entre toute paire de sommets adjacents dans G , il existe dans H un chemin de longueur ≤ 3 . Cela permettra de montrer deux choses : 1) H est un bien un *spanner*, c'est-à-dire tous les sommets sont couverts ; et 2) l'étirement de H est ≤ 3 , car si chaque arête de G est étirée par un facteur ≤ 3 , alors tout chemin de longueur d (en particulier un plus court chemin) sera également étiré par un facteur ≤ 3 .

Soit $xy \in E(G)$. Si $xy \in E(H)$, alors l'étirement de l'arête xy est 1. Supposons donc que $xy \notin E(H)$. La suppression de l'arête xy de H se produit seulement à l'étape 2(b). Soit u le sommet sélectionné à l'étape 2 et produisant la suppression de l'arête xy . Notons qu'à l'étape 2(a), après la sélection de u , l'arête xy existe encore dans G . Sans perte de généralité, on suppose que x est le sommet le plus proche de u dans G à ce moment là. On a $x \neq u$, car sinon xy serait ajoutée à l'étape 2(a). Le sommet x ne peut être à distance 2 de u , car sinon y serait à distance 2 ou 3, et l'arête xy ne serait pas supprimée à cette étape 2(b) : seulement $B_G(u, 1)$ est enlevée de G . Donc x est un voisin de u , et par voie de conséquence $y \in B_G(u, 2)$, toujours dans le graphe G du moment. L'étape 2(a) ajoute un chemin de longueur au plus deux entre u et tous ces sommets de $B_G(u, 2)$, en particulier vers y . À cette étape, l'arête xu est aussi ajoutée. Donc dans H il existe un chemin de longueur ≤ 3 entre x et y , c'est-à-dire $d_H(x, y) \leq 3$.

Montrons que la taille de H est au plus $n^{3/2} - n/2$. Soit $t \geq 0$ le nombre de fois où le test de l'étape 2 est vrai. On remarque qu'à l'étape 2(a) au plus $n - 1$ arêtes sont ajoutées. Après l'étape 2, on a donc ajouté $< tn$ arêtes à H et le nombre de sommets restant dans G est au plus $n - t \cdot (\lceil \sqrt{n} \rceil + 1) \leq n - t\sqrt{n} - t$. On a aussi bien évidemment $\deg(u) < \sqrt{n}$ pour les sommets u restant dans G . Le nombre d'arêtes de G à l'étape 3 est donc :

$$< \frac{1}{2}\sqrt{n} \cdot (n - t\sqrt{n} - t) = \frac{1}{2}n\sqrt{n} - \frac{1}{2}tn - \frac{1}{2}t\sqrt{n}.$$

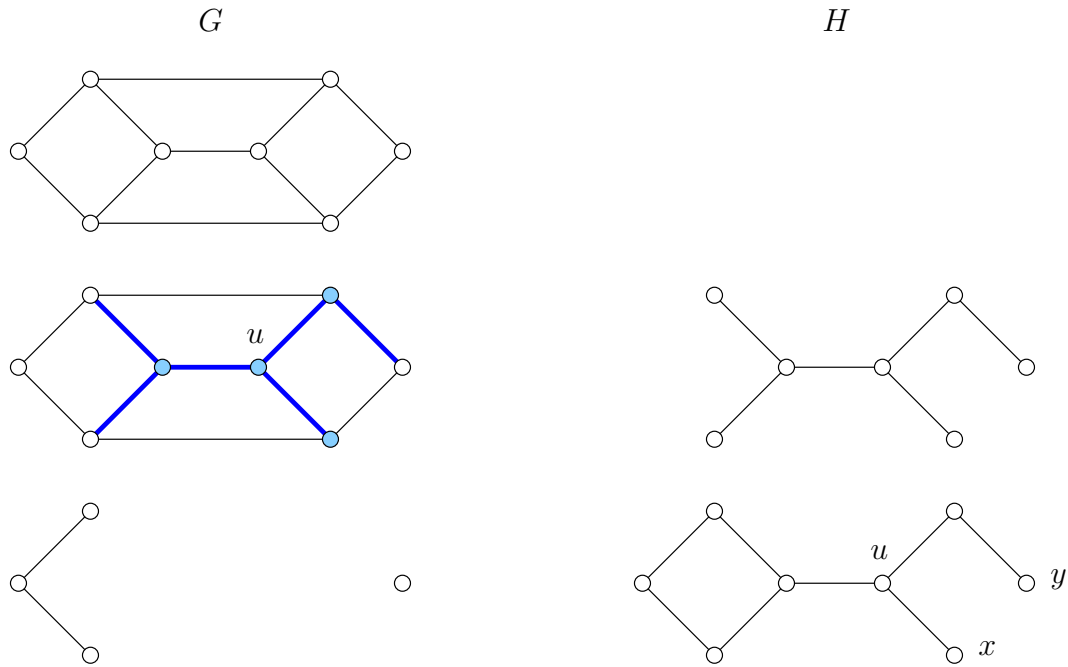


FIGURE 1.2 – Exemple d'exécution de l'algorithme $3\text{SPANNER}(G)$ avec $n = 8$ et $\sqrt{n} \approx 2.82$. L'exécution se lit en ligne de haut en bas. Au départ (ligne 1), H est vide. Le sommet u est sélectionné (ligne 2) et un arbre BFS de profondeur 2 est construit puis ajouté à H . Enfin, u est supprimé de G (ligne 3) et le graphe résultant est ajouté à H car G ne contient plus de sommet de degré $> \sqrt{n}$.

On a donc :

$$\begin{aligned}
 |E(H)| &< tn + \frac{1}{2}n\sqrt{n} - \frac{1}{2}tn - \frac{1}{2}t\sqrt{n} \\
 &< \frac{1}{2}n\sqrt{n} + tn - \frac{1}{2}tn - \frac{1}{2}t\sqrt{n} \\
 &= \frac{1}{2}n\sqrt{n} + \frac{1}{2}tn - \frac{1}{2}t\sqrt{n} \\
 &= \frac{1}{2}n\sqrt{n} + \frac{1}{2}t(n - \sqrt{n})
 \end{aligned}$$

Comme l'étape 2(b) supprime au moins $\sqrt{n}+1$ sommets, on en déduit que $t \leq n/(\sqrt{n}+1) < \sqrt{n}$. D'où,

$$|E(H)| < \frac{1}{2}n\sqrt{n} + \frac{1}{2}\sqrt{n}(n - \sqrt{n}) = n^{3/2} - \frac{1}{2}n .$$

Ceci qui termine la preuve. □

Remarquons que le théorème 1 devient faux si dans l'énoncé l'on remplace l'étirement 3 par toute valeur < 3 , et ce même si le nombre d'arêtes augmente significativement, par exemple de $n^{1.5}$ à $n^{1.9}$ ce qui autorise qu'en même $n^{0.4}$ fois plus d'arêtes. En effet, le graphe biparti complet à n sommets $K_{\lfloor n/2 \rfloor, \lceil n/2 \rceil}$ possède $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \approx n^2/4$ arêtes. Or si l'on enlève au moins une arête dans ce graphe, l'étirement du sous-graphe obtenu est au moins 3 car initialement tous les cycles sont de longueur au moins 4.

En fait le résultat précédent se généralise de la manière suivante. On retrouve, à $1.5n$ arêtes près, le théorème 1 en faisant $k = 2$.

Théorème 2 *Soit $k \geq 1$ un entier. Tout graphe à n sommets possède un $(2k - 1)$ -spanner de taille au plus $n^{1+1/k} + n$.*

Preuve. On considère l'algorithme suivant :

Algorithme SPANNER $_k(G)$

Entrée : un graphe G et un entier $k \geq 1$

Sortie : un *spanner* H

1. $H := (V(G), \emptyset)$
2. Pour chaque arête $xy \in E(G)$, si $d_H(x, y) > 2k - 1$, alors $H := H \cup \{xy\}$

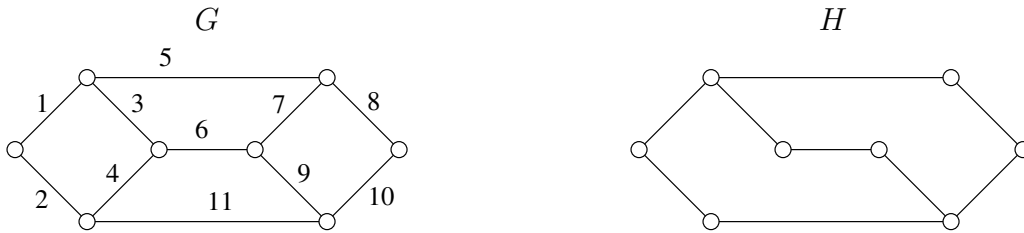


FIGURE 1.3 – Exemple d'exécution de l'algorithme SPANNER $_k(G)$ avec $k = 2$, en prenant les arêtes dans l'ordre de leur numéro. H ne contient plus de cycles de longueur 4, mais possède une arête de plus que le *spanner* produit par 3SPANNER(G) (cf. figure 1.2).

Pour l'analyse de l'étirement, comme précédemment, il suffit de montrer que pour chaque arête xy de G il existe un chemin dans H de x à y de longueur $\leq 2k - 1$. Soit xy une arête de G . Si $xy \in E(H)$, alors on a un chemin de longueur $1 \leq 2k - 1$. Supposons $xy \notin E(H)$. Dans l'algorithme, lorsqu'on décide de ne pas ajouter xy à H , on a $d_H(x, y) \leq 2k - 1$. C'est encore vrai dans le graphe H final. Donc il y a bien dans H un chemin de longueur $\leq 2k - 1$ entre x et y . L'étirement de H est $\leq 2k - 1$.

Avant de la démontrer, on va admettre la propriété importante suivante :

Proposition 1 *Soient H un graphe à n sommets et $k \geq 1$ un entier. Si H n'a pas de cycle de longueur $\leq 2k$, alors H possède $< n^{1+1/k} + n$ arêtes.*

Cette proposition permet de conclure quant à la taille de H , car il ne possède pas de cycle de longueur $\leq 2k$. En effet, on crée des cycles dans H seulement lors de l'ajout d'arêtes. Supposons qu'on est sur le point d'ajouter l'arête xy à H qui formera le plus petit cycle du H final. Juste avant l'ajout, on a $d_H(x, y) > 2k - 1$. Le plus court chemin de x à y dans H est donc de longueur au moins $2k$. Il suit que l'ajout de xy dans H forme un cycle de longueur au moins $2k + 1$: H ne peut avoir de cycle de longueur $\leq 2k$.

Il reste à démontrer la proposition.

Preuve de la proposition 1. Soit $d = m/n$ où m est le nombre d'arêtes de H . L'idée est de montrer que H contient un arbre complet de degré d et de profondeur k , et donc avec au moins $(d - 1)^k$ sommets. Du coup, on obtient l'inégalité $(d - 1)^k = (m/n - 1)^k < n$, ce qui implique notre résultat.

On construit un sous-graphe dense de H , noté D , en supprimant successivement les sommets de degré plus petit que d . Plus précisément, D est obtenu par la procédure suivante :

Algorithme DENSIFIE(H)

1. $D := H$
 2. Tant qu'il existe $u \in V(D)$, $\deg_D(u) < d$, faire $D := D \setminus \{u\}$.
-

Le graphe résultant $D := \text{DENSIFIE}(H)$ n'a pas de cycle de longueur $\leq 2k$, ces cycles étant au moins aussi long que ceux de H . Bien évidemment, tous les sommets de D , s'il y en a, sont de degré $\geq d$.

On va montrer que D possède au moins un sommet. Soit d_i le degré du sommet u sélectionné au début de la i -ème itération de l'étape 2 dans le graphe courant. Si D n'a pas de sommets, c'est que toutes les arêtes de H ont été enlevées en exécutant n fois l'étape 2, et donc que la somme $\sum_{i=1}^n d_i = m$. Or $d_i < d = m/n$. Donc cette somme est $< m$, ce qui est contradictoire. Donc D possède au moins un sommet. Il en possède même au moins $\lfloor d \rfloor + 2$ puisque ce sommet est de degré $\geq \lfloor d \rfloor + 1$.

Soit r un sommet de D et T un arbre en largeur d'abord (BFS) de racine r couvrant la composante connexe de D contenant r . Soit uv une arête de $D \setminus T$ telle que u, v sont tous les deux dans T . On note respectivement p_u, p_v leur profondeur dans T , ce qui correspond aussi à la distance dans D entre le sommet et r . Le point clef est que p_u et p_v sont $\geq k$.

En effet, on peut former dans D un cycle en utilisant T de r à u , puis T de v à r via l'arête uv . Sa longueur est $p_u + p_v + 1$. Or dans un arbre BFS, $|p_u - p_v| \leq 1$. Donc, en supposant que $p_u \leq p_v \leq p_u + 1$, on a dans D au mieux un cycle de longueur $2p_u + 2$. Cette longueur doit être $\geq 2k + 1$, D n'a pas de cycle plus court. Donc $p_u \geq k - 1/2$, ce qui implique $p_u \geq k$ car p_u est entier, et $p_v \geq k$ puisque $p_v \geq p_u$.

Ainsi, les seules arêtes de D qui ne sont pas dans T se trouvent entre sommets de profondeur k ou plus. Dit autrement, le sous-graphe de D induit par les sommets à distance $< k$ de r est un arbre. Il faut noter que dans D , tous les sommets ont un degré $> d$, c'est-à-dire $\geq \lfloor d \rfloor + 1$ puisque d n'est pas forcément un entier. C'est donc aussi vrai dans T pour

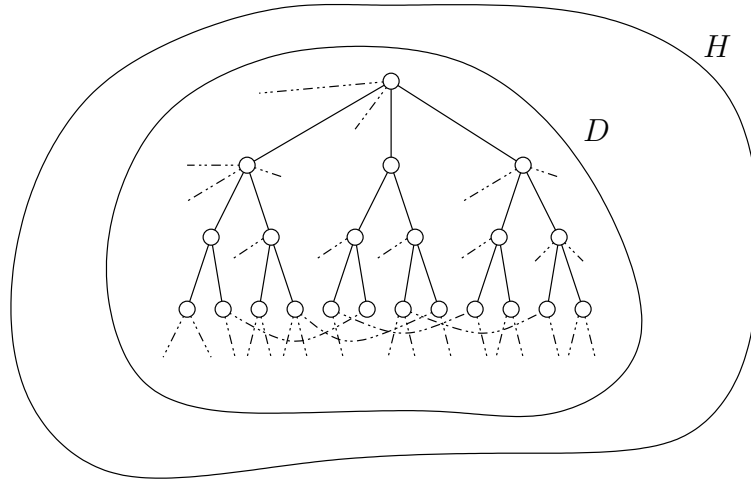


FIGURE 1.4 – Illustration de la preuve de la proposition 1. Le sous-graphe D de H contient un arbre complet de profondeur $k = 3$.

les sommets de profondeur $< k$: ils ont chacun $\geq \lfloor d \rfloor$ fils distincts (le “-1” vient du père). Ainsi, r a $\geq \lfloor d \rfloor + 1$ voisins de profondeur 1, chacun ayant $\geq \lfloor d \rfloor$ autres fils distincts, *etc.* Le nombre de sommets de D est donc au moins :

$$\begin{aligned} |V(D)| &\geq 1 + (\lfloor d \rfloor + 1) + (\lfloor d \rfloor + 1) \lfloor d \rfloor + (\lfloor d \rfloor + 1) \lfloor d \rfloor^2 + \dots + (\lfloor d \rfloor + 1) \lfloor d \rfloor^{k-1} \\ &> (\lfloor d \rfloor + 1) \lfloor d \rfloor^{k-1} > \lfloor d \rfloor^k = \left\lfloor \frac{m}{n} \right\rfloor^k > \left(\frac{m}{n} - 1 \right)^k. \end{aligned}$$

Bien sûr, $|V(D)| \leq n$, donc $(m/n - 1)^k < n$, ce qui implique $m < n^{1+1/k} + n$ et termine la preuve. \square

Ceci termine la preuve du théorème 2. \square

En fait, le résultat est valable même si H possède une arête-évaluation. Dans ce cas il faut lister les arêtes de H par coût croissant. On remarquera qu’en posant $k = +\infty$, l’algorithme $\text{SPANNER}_{+\infty}$ est celui de Kruskal : les arêtes de H sont ajoutées tant que H reste acyclique. H est alors une forêt (un arbre si H est connexe) couvrante de poids minimum (c’est-à-dire dont la somme des poids des arêtes est la plus petite possible).

On pourrait se demander pourquoi on a présenté l’algorithme 3SPANNER alors que SPANNER_k fait la même chose pour $k = 2$. En fait, contrairement au théorème 2, le résultat du théorème 1 peut être encore raffiné en remarquant que l’analyse du nombre d’arêtes est insensible à la profondeur de l’arbre (BFS) qui est appliqué dans l’algorithme 3SPANNER. L’utilisation d’un arbre couvrant tout le graphe, au lieu de seulement $B_G(u, 2)$, réduit considérablement l’étirement du *spanner*, comme on va le voir dans le théorème 3 ci-dessous.

On utilise la définition suivante :

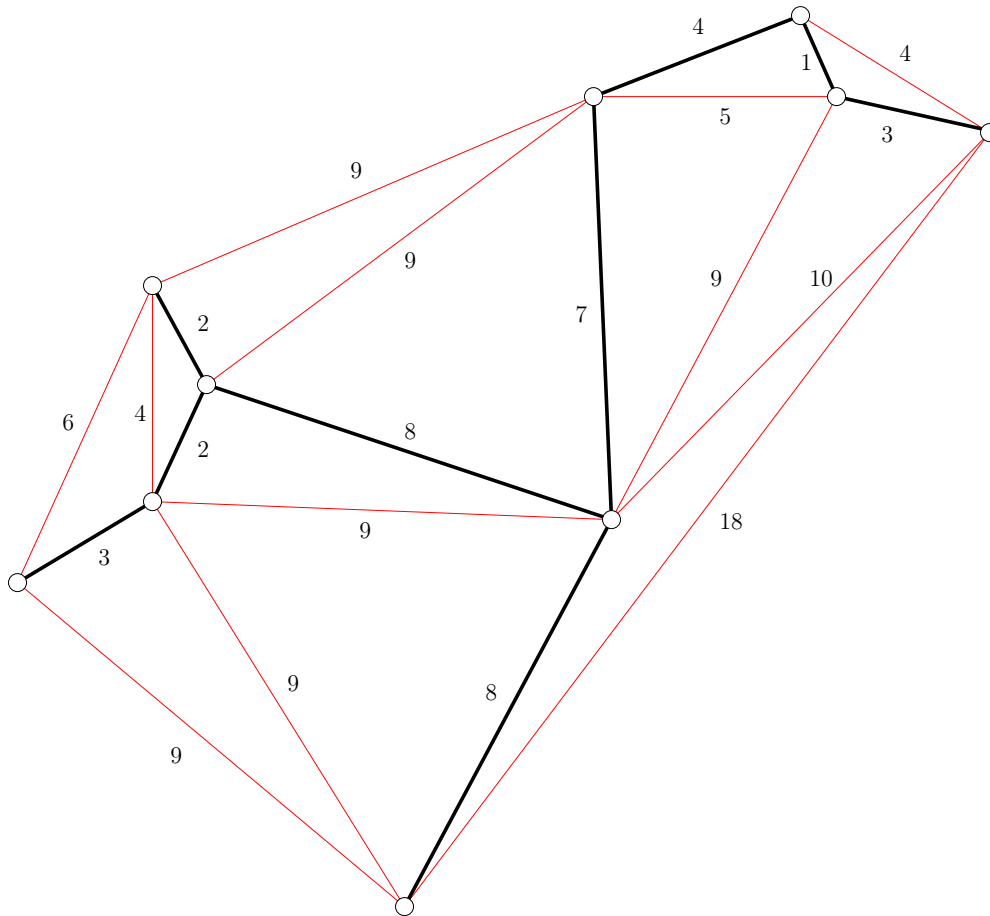


FIGURE 1.5 – Exemple d'arbre couvrant de poids minimum (en noir).

Définition 2 Une sous-graphe H d'un graphe G est un (α, β) -spanner si pour toute paire de sommets $x, y \in V(G)$, $d_H(x, y) \leq \alpha \cdot d_G(x, y) + \beta$.

Un α -spanner est donc un $(\alpha, 0)$ -spanner. On peut vérifier qu'un (α, β) -spanner est aussi un $(\alpha + \beta)$ -spanner. Le théorème 3 ci-dessous est donc plus « fort » que le théorème 1, puisqu'il l'implique : un $(1, 2)$ -spanner est un 3-spanner.

Théorème 3 Tout graphe à n sommets possède un $(1, 2)$ -spanner de taille au plus $n^{3/2} - n/2$.

Preuve. On considère l'algorithme 3SPANNER dans lequel l'instruction « $B_G(u, 2)$ » de l'étape 2(a) est remplacée par « $B_G(u, +\infty)$ ». On ne fait que l'analyse de l'étirement, l'analyse du nombre d'arêtes étant identique à celle donnée dans la preuve du théorème 1. (On avait borné par n le nombre d'arêtes ajoutées à l'étape 2(a), ce qui est encore vrai dans cette nouvelle version.)

Soit x, y deux sommets de G_0 , le graphe initial, et P un plus court chemin entre x et y avec $d = d_{G_0}(x, y)$. On va supposer qu'au moins une arête de P n'est pas dans H , sinon $d_H(x, y) = d_{G_0}(x, y)$. Des arêtes manquent à H à cause de l'étape 2(b) où l'on supprime du graphe courant G la boule $B_G(u, 1)$.

On note u le premier sommet sélectionné dont $B_G(u, 1)$ intersecte P . Ici G représente le graphe juste avant la suppression de $B_G(u, 1)$. Ainsi, P existe dans G , en particulier x et y existent dans G . Dans H , on ajoute un plus court chemin dans G (courant !) entre u et tous les autres restant, en particulier vers x et y . Donc, $d_H(x, y) \leq d_G(x, u) + d_G(u, y)$. Soit v un sommet de $B_G(u, 1) \cap P$. On a $d_G(x, u) \leq d_P(x, v) + 1$. De même, $d_G(u, y) \leq 1 + d_P(v, y)$. Donc, $d_H(x, y) \leq d_P(x, v) + d_P(v, y) + 2$. Or, $d_P(x, v) + d_P(v, y) = d_P(x, y) = d = d_{G_0}(x, y)$ car P est un plus court chemin. Donc, $d_H(x, y) \leq d + 2 = d_{G_0}(x, y) + 2$ ce qui montre que H est un $(1, 2)$ -spanner. \square

Notons ici que les arêtes doivent avoir un poids uniformes. Sinon, l'étirement additif est $\leq 2\Delta$, où Δ est l'*aspect ratio* de l'arête-valuation du graphe, c'est-à-dire le ratio du poids maximum sur le poids minimum d'une arête.

Malheureusement, ce théorème ne se généralise pas pour tous les $k > 2$ comme dans le théorème 2. Un résultat similaire existe cependant pour $k = 3$: tout graphe possède un $(1, 6)$ -spanner de taille $O(n^{4/3})$ (cf. [BKMP05]). Mais personne ne sait s'il existe ne serait-ce qu'une constante β produisant, pour tout graphe, un $(1, \beta)$ -spanner de taille $o(n^{4/3})$ (voir [Pet07] pour plus de détails).

Bibliographie

- [AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2) :781–793, 2004.
- [BKMP05] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β) -spanners and purely additive spanners. In *16th Symposium on Discrete Algorithms (SODA)*, pages 672–681. ACM-SIAM, January 2005.
- [Für09] Martin Fürer. Faster multiplication algorithm. *SIAM Journal on Computing*, 39(3) :979–1005, 2009.
- [Pet07] Seth Pettie. Low distortion spanners. In *34th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 of Lecture Notes in Computer Science, pages 78–89. Springer, July 2007.

2.1 Temps polynomial *vs.* exponentiel

Petit rappel sur la notation « O ». Il s'agit d'une convention d'écriture dont le sens est le suivant. Lorsque l'on écrit, par exemple, que $t(n) = 2^{O(f(n))}$, cela signifie que :

$$\exists c > 0, n_0, \quad \forall n \geq n_0, \quad t(n) \leq 2^{c \cdot f(n)}.$$

Il faut donc voir la notion « O » non pas comme une valeur ou fonction précise, mais plutôt comme un majorant. Par exemple, on peut écrire que $\sin n = O(1)$, où que la complexité d'un programme est en $\log^{O(1)} n$ pour dire qu'elle est polylogarithmique. C'est la même chose pour la notation « Ω » que l'on doit voir comme un minorant. La notation Θ est un majorant et minorant à la fois. On écrit ainsi $\sin n = \Theta(1)$ puisque $\sin n = O(1)$ et $\sin n = \Omega(1)$.

Il faut lire cette notation de gauche à droite seulement, et bien se méfier des multiples « O » qui peuvent apparaître : $O(A) = O(B)$ n'est pas pareil que $O(B) = O(A)$. Chaque notation « O » se réfère à des constantes n_0 et c différentes.

Pour illustrer les complexités polynomiales et exponentielles, considérons une instance de taille $n \approx 50$ (par exemple une grille 7×7), un ordinateur d'une puissance d'un milliard d'instructions par seconde (soit environ 1 GHz), et une série d'algorithmes de différentes complexités :

polynomiale : n^c		exponentielle : c^n	
complexité	temps	complexité	temps
n^2	25 μ s	1.5^n	6 min
n^3	1 ms	2^n	130 j
n^5	3 s	3^n	230 années

Il y a d'autres complexités, qui sont ni polynomiales ni exponentielles. Par exemple $n^{\log n + O(1)} = n^{O(1)} \cdot 2^{\log^2 n}$ pour résoudre l'isomorphisme de groupe à n éléments, algorithme due à Tarjan¹.

1. Dans ce problème, on a en entrée deux tables carrées $n \times n$ décrivant l'opération interne de chaque

Le problème des complexités exponentielles est qu'il ne suffit pas de gagner un ordre de grandeur ($\times 10$) sur la puissance de l'ordinateur pour améliorer définitivement la situation. Bien sûr, le problème de complexité 2^n sur un ordinateur 10 fois plus puissant s'exécutera en 13j au lieu de 130j.

Cependant, si l'on double la taille du problème, l'instance est disons un graphe à $n = 100$ sommets (une grille à 10×10), alors, même avec le nouvel ordinateur, le temps sera de :

$$T = 2^{2n} = (2^n)^2 \approx (13 \text{ j})^2 = 179 \text{ j}$$

et si l'on veut résoudre un problème 10 fois plus grand (une grille 33×33) :

$$T = 2^{10n} = (2^n)^{10} \approx (13 \text{ j})^{10} = 130 \text{ milliards d'années} .$$

2.2 Problèmes jouets

On va définir trois problèmes que l'on va suivre tout au long de ce cours. Commençons par quelques définitions (voir figure 2.1 pour une illustration) :

- COUVERTURE PAR SOMMETS (*Vertex Cover*) : ensemble de sommets contenant au moins une extrémité de chacune des arêtes. En général, on cherche une couverture par sommets de la plus petite taille possible.
- ENSEMBLE INDÉPENDANT (*Independent Set*) : ensemble de sommets deux à deux non adjacents. Dans le graphe complémentaire, c'est une clique. En général, on cherche un ensemble indépendant de la plus grande taille possible.
- ENSEMBLE DOMINANT (*Dominating Set*) : ensemble de sommets S tel que tout sommet $u \notin S$ a au moins un voisin dans S . En général, on cherche un ensemble dominant de la plus petite taille possible.

Les trois problèmes de décisions associés sont : COUVERTURE PAR SOMMETS de taille k , ENSEMBLE INDÉPENDANT de taille k et ENSEMBLE DOMINANT de taille k où le but est de savoir si un graphe G possède une couverture par sommets (resp. ensemble indépendant, ensemble dominant) de taille k .

Ces trois problèmes sont NP-complets, même pour les graphes planaires. L'intérêt de ces problèmes est qu'ils sont à la fois très simple (cas d'école) et centraux en théorie de la complexité car beaucoup de problèmes NP-complets se réduisent facilement à ceux-là.

On remarquera qu'une couverture par sommets est un ensemble dominant, le contraire étant faux en général. Également, le complémentaire d'une couverture par sommets est un ensemble indépendant, et qu'inversement, le complémentaire d'un ensemble indépendant

groupe – par exemple la table d'addition – et il faut déterminer si elles sont isomorphes. Il y a *a priori* $n! \cdot n^2$ vérifications à faire. Dans le cas de groupe Abélien (c'est-à-dire commutatif), un algorithme en $O(n \log n)$ existe [Vik96], ce qui est moins que le nombre d'entrées de la table !

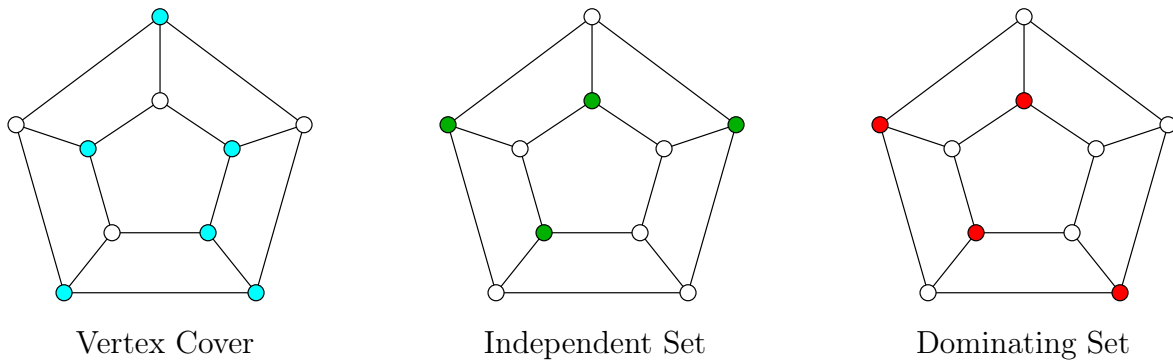


FIGURE 2.1 – Définition des problèmes jouets.

est une couverture par sommets. [Exercice : pourquoi ?] Bien sûr, ENSEMBLE INDÉPENDANT est équivalent au problème CLIQUE dans le graphe complémentaire.

2.3 Algorithmes exhaustifs (*brute force*)

Technique consistant à lister toutes les solutions possibles et à vérifier chacune d'elle.

Pour les problèmes de type « sous-ensemble de sommets d'un graphe G », comme les trois problèmes jouets précédant, cela donne :

$\forall S \subseteq V(G)$ de taille k , vérifier si S est du type recherché.

Vérifier que S est une couverture par sommets, un ensemble indépendant ou un ensemble dominant, peut se faire en temps $O(n + m) = O(n^2)$ où $n = |V(G)|$ et $m = |E(G)|$. Donc pour savoir s'il existe un ensemble S désiré de taille k il faut un temps :

$$T = \binom{n}{k} \cdot O(n^2) = O(n^{k+2}) \quad \text{car}$$

$$\binom{n}{k} = \mathfrak{C}_n^k = \frac{n!}{k!(n-k)!} < n \cdot (n-1) \cdots (n-k+1) < n^k.$$

Cette méthode est appliquée lorsqu'on ne sait rien faire d'autre. Elle peut *a priori* s'appliquer à tout problème qui est dans NP, car chacun de ces problèmes admet un certificat et un vérificateur positif de complexité polynomiale (voir la section 3.1 au chapitre 3). Il suffit donc de lister tous les certificats positifs possibles et d'appliquer le vérificateur. Ceci dit, il y a des problèmes prouvés dans P où le vérificateur peut être très difficile à obtenir, comme par exemple les problèmes définis à partir d'une liste finie de mineurs exclus (voir la section 2.7). La liste peut être finie mais pas connue de manière explicite, comme la liste des mineurs exclus minimaux pour les graphes toriques (c'est-à-dire dessinable sur le tore).

Cette méthode peut donner de bon résultats seulement si k est très petit ($k \leq 5$) et n pas trop grand ($n \leq 50$). Une complexité en $n^{k+1} = n^6$ à 1 GHz prend déjà 30 minutes pour $n = 50$.

Pour comprendre le phénomène d'explosion combinatoire, imaginons que l'on souhaite trier un jeu de cartes en appliquant la méthode exhaustive, et donc en ignorant les algorithmes polynomiaux bien connus. La méthode consiste donc ici à mélanger les cartes pour tous les ordres possibles et à vérifier (en temps linéaire) que le tas est trié. Cet algorithme est de complexité $n! \cdot n \approx (n/e)^{n+1}$, n étant le nombre de cartes, tout comme l'algorithme très naïf du voyageur de commerce à n villes.

Supposons maintenant que cet algorithme est implanté sur une architecture parallèle de processeurs surpuissants, chacun capable de tester un milliard de mélange par seconde, associant un tel processeur par particule de l'Univers² ($\approx 2^{400}$), et qui est exécuté pendant toute la durée de vie de l'Univers (< 15 Ma). Combien, dans ces conditions, pourrait-on trier de cartes? Réponse : à peu près 52.

2.4 Complexité paramétrique : définition

L'idée est de paramétrer le problème et de produire des algorithmes spécifiques et optimisés pour ce paramètre. Par exemple, on peut être intéressé de produire des algorithmes optimisés en fonction du *genre* du graphe, c'est-à-dire du nombre de trous de la surface sur lequel est dessiné le graphe (0 trou pour les graphes planaires, 1 trou pour les graphes toriques, ...).

Dans la pratique n est très grand ($n > 10^6$) et on espère que le paramètre soit petit. Typiquement, la triangulation surfacique d'objets réels (une carrosserie de voiture ou d'avion par exemple) est un graphe ou maillage comprenant de nombreux sommets ($n \approx \#\text{triangles}$) alors que le nombre de trous de la surface supportant le graphe est limité (fenêtres, portes, *etc.*). Par ce biais, on espère que la difficulté du problème est liée au fait que le paramètre k soit grand, et non pas n (cf. figure 2.2).

Définition 3 *Un problème Π de paramètre k est de complexité paramétrique polynomiale (en abrégé FPT pour Fixed Parametrized Tractable) si la complexité en temps de Π pour une instance de taille n est bornée par $f(k) \cdot n^{O(1)}$ pour une certaine fonction f .*

Supposons que Π soit un problème NP-complet et que k soit un paramètre borné par une fonction polynomiale en n , c'est-à-dire $k \leq n^{O(1)}$. Par exemple, si k représente la taille d'une couverture par sommets, d'un ensemble indépendant ou encore d'un ensemble dominant, on a $k \leq n$.

2. On estime à 10^{80} le nombre d'atomes de l'Univers. 2^{400} inclu toutes les particules élémentaires, quarks, *etc.*

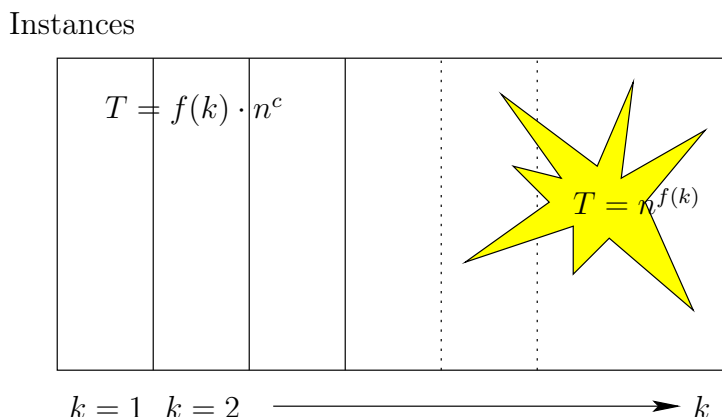


FIGURE 2.2 – Confinement de l’explosion combinatoire grâce à la paramétrisation.

Alors, si Π est FPT pour le paramètre k , la fonction f est très certainement exponentielle en k . Car sinon $f(k) \leq k^{O(1)}$ impliquerait que $f(k) \cdot n^{O(1)} \leq (n^{O(1)})^{O(1)} \cdot n^{O(1)} \leq n^{O(1)}$ ce qui n’est pas possible sauf si $P=NP$.

On peut démontrer qu’il existe, sauf si $P=NP$, des problèmes FPT en k où $f(k)$ n’est bornée par aucune pile d’exponentielles, c’est-à-dire

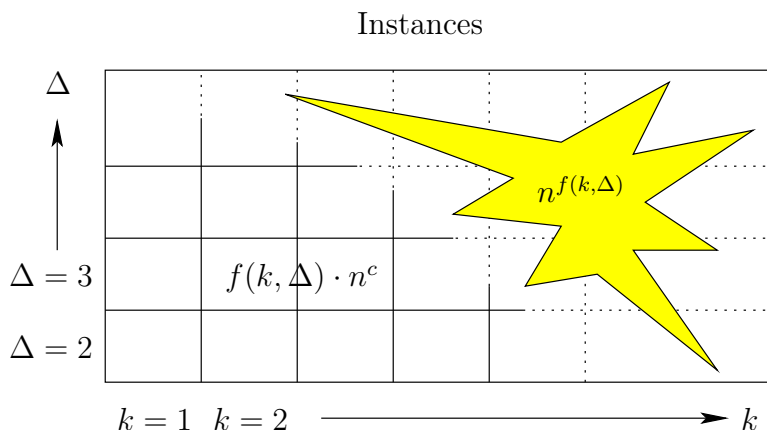
$$f(k) > 2^{k^{\dots^k}}.$$

Certains problèmes sont FPT suivant un paramètre et pas suivant d’autres. En voici quelques exemples :

<p>ENSEMBLE DOMINANT :</p> <p>Instance: un graphe G et un entier k</p> <p>Paramètre: k</p> <p>Question: est-ce que G possède un ensemble dominant de taille k ?</p>

A l’heure actuelle, on ne sait pas si le problème ci-dessus est FPT pour le paramètre k . La meilleure borne connue pour ce problème est $O(n^{k+1})$, et $O(2^{0.598n})$ pour trouver un ensemble dominant de taille minimum [FGK09]. Cependant, comme on le verra dans la suite du cours, le problème ENSEMBLE DOMINANT est FPT pour le paramètre $(k, \Delta(G))$ où $\Delta(G)$ est le degré maximum du graphe. Dans ce cas la complexité est en $O((\Delta(G)+1)^k \cdot (n+m))$. Dit autrement, le problème a une complexité polynomiale lorsque k et $\Delta(G)$ sont bornés (cf. figure 2.3).

Et pour le problème de la k -coloration ? c’est-à-dire peut-t-on colorier les sommets d’un graphe en utilisant au plus k couleurs sans avoir deux sommets voisins de la même couleur ?

FIGURE 2.3 – Paramétrage pour la k -coloration.

k -COLORATION :

Instance: un graphe G et un entier k

Paramètre: k

Question: est-ce que G possède une k -coloration ?

La k -COLORATION n'est certainement pas FPT en k (sauf si $P=NP$ ou $k=2$) puisque sinon cela impliquerait que la 3-COLORATION serait polynomiale. Or c'est un problème NP-complet. Cependant, on verra dans la section 2.8 que le même problème, mais paramétré différemment, est FPT en la *largeur arborescente* du graphe. La largeur arborescente de G est notée $tw(G)$ pour *treewidth*. Le problème suivant est donc FPT :

k -COLORATION :

Instance: un graphe G et un entier k

Paramètre: $tw(G)$

Question: est-ce que G possède une k -coloration ?

Comme ENSEMBLE DOMINANT de taille k , on pense que CLIQUE de taille k n'est pas FPT en k . Cependant la situation est un peu meilleure que pour ENSEMBLE DOMINANT. Le meilleur algorithme connu pour CLIQUE (ou ENSEMBLE INDÉPENDANT) de taille k (Něsetřil et Poljak 1985 [NP85]) est³ $O(n^{2.38k/3}) = O(n^{0.79k})$. Comme on va le voir COUVERTURE PAR SOMMETS de taille k est FPT en k , ce qui laisse à penser que les trois problèmes s'ordonnent d'après leur complexité selon l'ordre :

3. Le 2.38 est l'exposant du meilleur algorithme de multiplication de matrices $n \times n$ (Coppersmith et Winograd 1990).

COUVERTURE PAR SOMMETS \prec ENSEMBLE INDÉPENDANT \prec ENSEMBLE DOMINANT .

Remarque : Pour 3-SATISFIABILITÉ, il existe un algorithme en $O(1.49^n)$ où n est le nombre de variables de la formule (l'algorithme naïf étant en $\Omega(2^n)$ consistant à vérifier toutes les affectations possibles des variables).

2.5 Arbre borné de recherche

Idée est de décomposer l'algorithme en deux parties :

1. Construire, peut être de manière inefficace, un espace de recherche qui est souvent un arbre de recherche de taille exponentielle.
2. Appliquer ensuite un algorithme assez efficace sur chaque nœud de l'arbre, souvent basé sur un parcours simple de l'arbre.

Le point crucial est que pour certains problèmes la taille de l'arbre ne dépend que du paramètre. Donc l'espace de recherche devient borné pour un paramètre k fixé.

Remarque : le problème de trouver un ensemble S « de taille k » ou le problème de trouver un ensemble S « de taille $\leq k$ » (resp. « de taille $\geq k$ ») sont identiques tant que que $k \in \{0, \dots, n\}$ et qu'on a la propriété que si S est une solution alors, pour tout sommet x , $S \cup \{x\}$ en est aussi une (resp. $S \setminus \{x\}$). C'est bien le cas des trois problèmes jouets.

2.5.1 COUVERTURE PAR SOMMETS de taille k

Théorème 4 (1992) COUVERTURE PAR SOMMETS de taille k pour les graphes à n sommets et m arêtes peut être résolu en temps $O(2^k \cdot (n + m))$.

Ce problème est donc FPT en k .

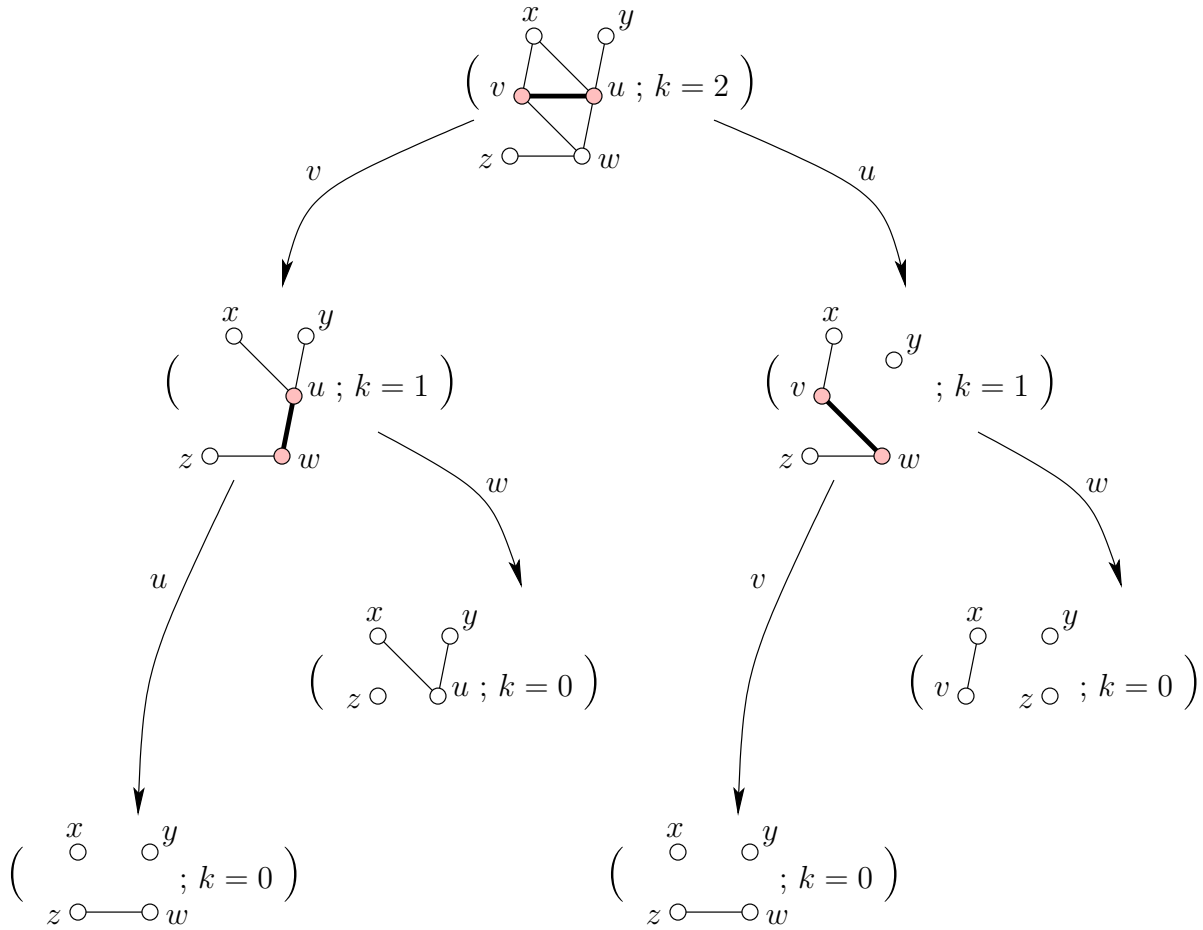
Preuve. On considère l'algorithme suivant :

_____ Algorithme VC1(G, k) _____

Entrée : un graphe G et un entier k

Sortie : VRAI si et seulement si G possède une couverture par sommets de taille $\leq k$.

1. Si $E(G) = \emptyset$, renvoyer VRAI.
2. Si $k = 0$, renvoyer FAUX.
3. Choisir arbitrairement une arête $\{u, v\}$ de G .
4. Construire $G_1 := G \setminus \{u\}$ et $G_2 := G \setminus \{v\}$.
5. Renvoyer $\text{VC1}(G_1, k - 1) \vee \text{VC1}(G_2, k - 1)$.

FIGURE 2.4 – Arbre de recherche à 7 nœuds lors d'un appel à l'algorithme $VC1(G, 2)$.

Montrons que l'algorithme est valide. Ce qui n'est pas immédiatement trivial c'est que les choix arbitraires de la ligne 3 mène toujours à la bonne réponse. On pourrait s'attendre à ce qu'un algorithme correct soit amené à vérifier tous les choix de sommets possibles, pour être sûr de ne rater aucune solution. On va le voir : l'espace des solutions possibles (l'arbre) est borné par une fonction du paramètre k .

L'algorithme est clairement correct si G n'a pas d'arête (ligne 1) ou si $k = 0$ (ligne 2). Ensuite, par induction, supposons que $VC1(H, k - 1)$ est correct pour tout graphe H . Il y a deux cas : G possède ou ne possède pas de couverture par sommets de taille $\leq k$. Considérons une arête quelconque de G , disons $\{u, v\}$.

Si G possède une couverture C de taille $\leq k$, alors soit $u \in C$ soit $v \in C$ (soit les deux). Si $u \in C$, alors $C \setminus \{u\}$ est une couverture par sommets pour $G \setminus \{u\}$. (On utilise ici le fait qu'une couverture par sommets reste valide si on supprime un sommet du graphe.) Sa

taille est $\leq k - 1$ si bien que $\text{VC1}(G_1, k - 1)$ est VRAI. Il en va de même si $v \in C$. Et donc $\text{VC1}(G_1, k - 1)$ ou $\text{VC1}(G_2, k - 1)$ est VRAI.

Si G ne possède pas de couverture par sommets de taille $\leq k$, alors $G \setminus \{u\}$ et $G \setminus \{v\}$ ne peuvent avoir de couverture par sommets de taille $k - 1$, sinon l'ajout de u ou de v constituerait une couverture de taille $\leq k$ pour G . Autrement dit, $\text{VC1}(G_1, k - 1)$ et $\text{VC1}(G_2, k - 1)$ sont FAUX tous les deux.

Dans les deux cas la ligne 5 est justifiée.

Calculons la complexité de $\text{VC1}(G, k)$. L'instruction la plus coûteuse (en dehors des appels récurifs) est l'instruction 4, qui prend un temps $O(n + m)$. Soit a_k le nombre maximum de fois que cette instruction est exécutée lors d'un appel à $\text{VC1}(G, k)$. À cause de l'évaluation paresseuse, il est possible que seul un branchement soit évalué (si le résultat est VRAI). On a donc une complexité totale pour l'algorithme de $O(a_k \cdot (n + m))$.

On a $a_0 = 0$, $a_1 = 1$, et de manière plus générale, $a_k \leq 1 + 2a_{k-1}$ pour tout $k > 0$. (En fait $a_k = 1 + 2a_{k-1}$) Il suit :

$$\begin{aligned} a_k &\leq 1 + 2(1 + 2a_{k-2}) = 2^0 + 2^1 + 2^2 a_{k-2} \\ &\leq 2^0 + \dots + 2^{i-1} + 2^i a_{k-i} \quad (\text{pour tout } i \geq 1) \\ &\leq 2^0 + \dots + 2^{k-1} + 2^k a_0 \quad (\text{pour } i = k) \\ &\leq 2^0 + \dots + 2^{k-1} = 2^k - 1 \end{aligned}$$

D'où au total une complexité de $O(2^k \cdot (n + m))$. □

Pour s'apercevoir que le graphe de la figure 2.1 n'a pas de couverture de taille < 6 , nous aurions la méthode exhaustive du effectuer $\binom{n}{k} = 10!/5!^2 = 252$ tests. L'algorithme VC1 nécessite seulement $2^5 - 1 = 31$ tests. En quelques sortes, l'arbre fourni un certificat, négatif ici, pour $k < 6$.

2.5.2 ENSEMBLE INDÉPENDANT pour les graphes planaires

Rappelons que, dans le cas général, ENSEMBLE INDÉPENDANT n'est pas connu pour être FPT en k , la meilleure complexité connue est celle de CLIQUE qui est $O(n^{0.79k})$. On restreint donc le problème à certaines instances, les graphes planaires où le problème reste NP-complet.

Théorème 5 ENSEMBLE INDÉPENDANT de taille k pour les graphes planaires à n sommets peut être résolu en temps $O(6^k \cdot n)$.

Ce problème est donc FPT en k . En fait, comme on va le voir dans la preuve du théorème 5, on n'utilise pas vraiment la planarité du graphe, c'est-à-dire le fait de pouvoir dessiner le graphe dans le plan sans croisement d'arêtes, mais plutôt son faible nombre

d'arêtes. N'analyse de l'algorithme donné dans la preuve se généralise à tout graphe t -dégénéré, et la complexité en temps pour résoudre ENSEMBLE INDÉPENDANT de taille k est de $O((t+1)^k \cdot tn)$.

On rappelle qu'un graphe t -dégénéré est un graphe où tout sous-graphe induit possède un sommet de degré $\leq t$. Ces graphes peuvent ainsi être « épluchés » en enlevant successivement un sommet de faible degré.

La formule d'Euler ($\chi = m - n + f = 2$) implique que :

Fait 1 *Tout graphe planaire à $n \geq 3$ sommets possède au plus $3n - 6$ arêtes.*

Comme corollaire, on obtient que tout graphe planaire possède un sommet de degré ≤ 5 . Les graphes planaires sont donc 5-dégénérés. En effet, $\sum_{u \in V(G)} \deg(u) = 2m$, m est le nombre d'arêtes de G . Donc il existe un terme de cette somme, le degré d'un certain sommet, $\leq \frac{1}{n} \sum_u \deg(u) = 2m/n < 6$ d'après le fait 1.

Le même argument permet de montrer que les arbres sont 1-dégénérés car $m = n - 1$ dans ce cas.

Preuve. On considère le programme suivant, où le paramètre $t = 5$ pour les graphes planaires (on rappelle que $B(u, r)$ est la boule de rayon r centrée en u , cf. chapitre 1) :

Algorithme $\text{EI}_t(G, k)$

Entrée : un graphe t -dégénéré G et un entier k

Sortie : VRAI si et seulement si G possède un ENSEMBLE INDÉPENDANT de taille k .

1. Si $k > |V(G)|$, renvoyer FAUX.
 2. Si $E(G) = \emptyset$ ou si $k = 0$, renvoyer VRAI.
 3. Choisir un sommet u_0 de degré $d \leq t$ avec pour voisins u_1, \dots, u_d .
 4. Pour tout $i \in \{0, \dots, d\}$, construire le graphe $G_i := G \setminus B(u_i, 1)$.
 5. Renvoyer $\bigvee_{i=0}^d \text{EI}_t(G_i, k - 1)$.
-

À titre d'exemple, exécutons cet algorithme sur une étoile à 6 branches avec $k = 7$ et $t = 5$. Les tests des lignes 1 et 2 ne s'appliquent pas. Puis la ligne 3 sélectionne un sommet u_0 qui est une feuille. On construit en ligne 4 deux graphes : G_0 composés de 5 sommets isolés, et G_1 qui est vide. Ensuite, $\text{EI}_t(G_0, k - 1)$ et $\text{EI}_t(G_1, k - 1)$ sont évalués à FAUX tous les deux à cause de la ligne 1, $k - 1 = 6$. Et donc $\text{EI}_t(G, k) = \text{FAUX}$, ce qui est correct.

Remarque : En pratique, on a intérêt à la ligne 3 de choisir un sommet de degré le plus faible possible, l'arbre des solutions dépendant directement du degré des sommets ainsi sélectionnés.

Montrons que l'algorithme est correct. L'algorithme est correct pour $k = 0$. Montrons qu'il est aussi pour tout $k > 0$ en supposant que $\text{EI}_t(H, k - 1)$ est correct pour tout graphe H . Soit u_0 un sommet de G ayant pour voisin u_1, \dots, u_d . Il y a deux cas :

Supposons que G possède un ensemble indépendant de taille k . Dans ce cas, il un ensemble indépendant J de taille k contenant l'un des u_i . En effet, si I est un ensemble indépendant de taille k contenant aucun u_i , on peut ajouter u_0 et construire ainsi un nouvel ensemble indépendant pour G , aucun des voisins u_0 n'étant dans I . On obtient un ensemble de taille exactement k en enlevant un sommet quelconque de I .

Soit i un indice tel que $u_i \in J$. On a alors que $J \cap V(G_i)$ est un ensemble indépendant de G_i . Sa taille est $k - 1$, car comme $u_i \in J$, aucun voisin de u_i n'est dans J et donc tous les sommets autres que u_i sont bien dans J . Il suit que $\text{EI}_t(G_i, k - 1)$ est VRAI, et donc $\text{EI}_t(G, k)$ sera évalué à VRAI, ce qui est correct.

Supposons que G ne possède pas d'ensemble indépendant de taille k . Alors pour tout sommet u_i , G_i ne peut posséder d'ensemble indépendant de taille $k - 1$, puisque sinon G posséderait un ensemble indépendant de taille k en prenant celui de taille $k - 1$ pour G_i et en ajoutant u_i . Autrement dit $\text{EI}_t(G_i, k - 1)$ est FAUX pour tous les u_i , et donc $\text{EI}_t(G, k)$ sera évalué à FAUX, ce qui est correct.

Donc dans tous les cas $\text{EI}_t(G, k)$ est correct.

Complexité : L'opération la plus coûteuse, à part les appels récursifs, est la ligne 4, qui prend un temps $(d + 1) \times O(n + m) = O(tn + tm) = O(t^2 \cdot n)$ car le nombre d'arêtes est $m \leq tn$ pour un graphe t -dégénéré.

Soit a_k le nombre maximum de fois que la ligne 4 est exécutée pour un appel à $\text{EI}_t(G, k)$ on a alors que $a_k \leq 1 + t \cdot a_{k-1}$ avec $a_0 = 0$. Cela implique que

$$a_k \leq (t + 1)^0 + (t + 1)^1 + \dots + (t + 1)^k a_0 = \sum_{i=0}^{k-1} (t + 1)^i = \frac{((t + 1)^k - 1)}{t} < \frac{(t + 1)^k}{t}.$$

Donc la complexité totale est $O(a_k \cdot t^2 \cdot n) = O(t(t + 1)^k \cdot n)$, soit $O(6^k \cdot n)$ pour les graphes planaires. \square

2.5.3 ENSEMBLE DOMINANT pour les graphes planaires

Rappelons que, dans le cas général, ENSEMBLE DOMINANT n'est pas connu pour être FPT en k . La meilleure complexité connue est en $O(n^{k+1})$. On restreint donc le problème à certaines instances, les graphes planaires où le problème reste NP-complet. Cela sera aussi l'occasion de voir la technique de réduction de données.

ENSEMBLE DOMINANT est un problème plus « complexe » que COUVERTURE PAR SOMMETS ou ENSEMBLE INDÉPENDANT car il n'est pas stable par sous-graphes : si S est une couverture par sommets pour G , et que H est un sous-graphe de G (pas forcément induit), alors la restriction de S aux sommets de H , c'est-à-dire $S \cap V(H)$, est aussi une couverture par sommets pour H (on s'en sert pour la preuve du théorème 4). Il en va de même pour un ensemble indépendant S (cf. preuve du théorème 5). C'est malheureusement faux pour un ensemble dominant S comme le montre l'exemple suivant (figure 2.5) :

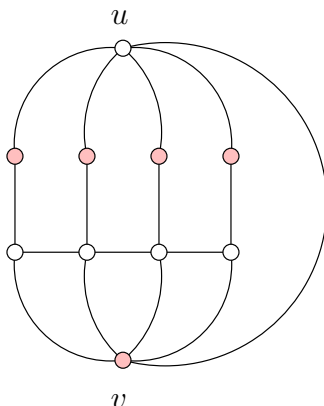


FIGURE 2.5 – $G \setminus \{u\}$ n'a pas d'ENSEMBLE DOMINANT de taille 1. De plus, tout ENSEMBLE DOMINANT de taille 2 contient u .

Intuitivement on a besoin de garder en mémoire le fait que certains sommets sont déjà dominés. Pour les appels récursifs, on ne peut ni les garder tels quels, ni les supprimer.

Dans l'exemple précédent (figure 2.5), en considérant la suppression de u : si on garde $N(u) = B(u, 1) \setminus \{u\}$ (les voisins de u , les sommets colorés), alors le graphe restant n'a pas d'ensemble dominant de taille 1. Si on enlève $N(u)$, il reste un chemin de longueur 4 qui n'a pas non plus d'ensemble dominant de taille 1.

Notons aussi que prendre le sommet de degré le plus élevé, ce qui est tentant, peut se révéler un mauvais choix – c'est cependant une bonne heuristique comme nous le verrons au chapitre 3.4 –. Prenons un graphe où un sommet u a pour voisinage un chemin $v_1 - v_2 - \dots - v_{3t}$, $t \geq 2$ entier, et dans lequel on ajoute un sommet de degré 1 à v_{3i+2} pour tout $i \in \{0, \dots, t-1\}$ (voir figure 2.6 où $t = 3$). Alors, la solution optimale (de taille t), et il y en a une, consiste à choisir les sommets v_{3i+2} de degré 4. Choisir u , de degré $3t \geq 6$, est donc un mauvais choix. Le même phénomène se produit avec un $K_{1,t}$ où chaque arête est subdivisée en deux.

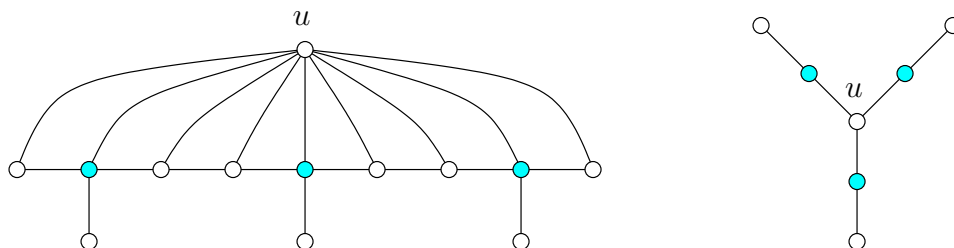


FIGURE 2.6 – Prendre le sommet de degré maximum n'est pas optimal pour ENSEMBLE DOMINANT.

On va donc résoudre un problème un petit peu plus général où le graphe d'entrée est

un graphe bicolorié : les sommets sont soit blancs soit noirs. Plus formellement :

ENSEMBLE DOMINANT ANNOTÉ :

Instance: un graphe G bicolorié et un entier k

Question: est-ce que G possède un ensemble de sommets S de taille k tel que tout sommet noir est soit dans S soit a un voisin dans S ?

Il s'agit donc de « dominer » seulement les sommets noirs. Évidemment, pour résoudre ENSEMBLE DOMINANT, on résout ENSEMBLE DOMINANT ANNOTÉ sur un graphe où tous les sommets sont initialement noirs.

Théorème 6 ([AFF⁺05]) ENSEMBLE DOMINANT ANNOTÉ de taille k pour les graphes planaires à n sommets peut être résolu en temps $O(8^k \cdot n^2)$.

Preuve. Comme annoncé, les sommets blancs codent des sommets déjà dominés. À chaque fois qu'on décidera de placer u dans un ensemble dominant pour un graphe bicolorié G , on supprimera u de G et on recoloriera tous ses voisins en blancs.

L'idée générale est assez simple : pour tout sommet u_0 qui est à dominer, soit u_0 est dans l'ensemble dominant, soit un de ses voisins u_1, \dots, u_d doit l'être. Sinon on viole la définition d'ensemble dominant. Ainsi, pour chaque $i \in \{0, \dots, d\}$, en supprimant u_i de G et blanchissant ses voisins, on obtient un algorithme dont on peut facilement vérifier la validité [Exercice].

La complexité de cet algorithme est contrôlée par le degré des sommets u_0 sélectionnés. Choisir un sommet u_0 de faible degré n'est pas suffisant, car u_0 n'est pas forcément un sommet qui est à dominer (noir). Et dans ce cas, on va produire un branchement dans l'arbre des solutions possibles sans pouvoir diminuer la taille du paramètre k . La taille de l'arbre n'est alors plus borné par une certaine fonction $f(k)$: on va tout droit vers une complexité exponentielle en n .

Ceci peut se produire en blanchissant d'abord $\Omega(n)$ sommets de degré ≤ 5 , si bien que les sommets noirs restant finissent par tous être de degré > 5 . Cela se produit avec le graphe de la figure 2.7 où les sommets blancs ont été générés par des sommets de degré 1 qui ont été d'abord supprimés.

Pour remédier à ce problème, on va combiner à la technique de l'arbre de recherche la technique de « réduction de données ». On réduit une instance (G, k) en une autre (G', k') en appliquant des règles, et ceci jusqu'à ce plus aucune des règles ne s'appliquent ou alors que $k = 0$. L'ordre d'application n'a pas d'importance.

Règles de réduction :

R1 : Supprimer les arêtes entre deux sommets blancs.

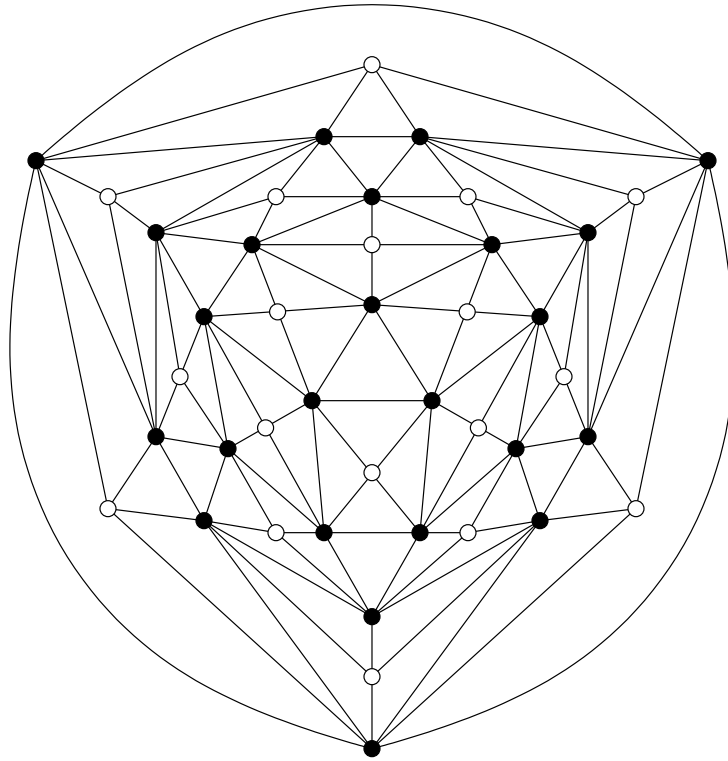


FIGURE 2.7 – Graphe bicolorié après réduction où tous les sommets noirs sont de degré 7.

R2 : Supprimer un sommet blanc v dont le voisinage est noir et est inclus dans le voisinage au sens large (boule) d'un sommet $u \neq v$ (cf. la figure 2.8).

R3 : Sélectionner le voisin v d'un sommet noir de degré 1, c'est-à-dire supprimer v du graphe, blanchir son voisinage, et diminuer d'un le paramètre k .

On ajoute parfois la règle suivante :

R4 : Si u est un sommet isolé, supprimer u et diminuer d'1 k si u est noir.

Cependant, elle n'est pas nécessaire car de fait, elle est appliquée par l'algorithme DSPA(G, k) ci-après.

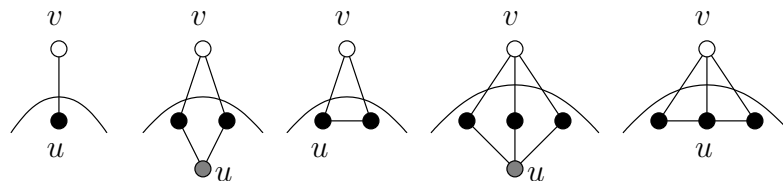


FIGURE 2.8 – Règle R2 limitée aux sommets v de degré ≤ 3 . Les sommets grisés peuvent être arbitrairement blancs ou noirs.

Dans la suite, on note $R_i(G, k)$, pour tout $i \in \{1, 2, 3\}$, la nouvelle instance (G', k')

obtenue en appliquant sur l'instance (G, k) la règle de réduction Ri .

Algorithme DSPA(G, k)

Entrée : un graphe G bicolorié et un entier k

Sortie : VRAI si et seulement si G possède un ENSEMBLE DOMINANT ANNOTÉ de taille $\leq k$.

1. Tant que $k > 0$ et qu'une règle Ri s'applique faire $(G, k) := Ri(G, k)$.
 2. Si G n'a pas de sommets noirs, renvoyer VRAI.
 3. Si $k = 0$, renvoyer FAUX.
 4. Choisir un sommet noir u_0 de degré $d \leq 7$ avec pour voisins u_1, \dots, u_d .
 5. Pour tout $i \in \{0, \dots, d\}$, construire $G_i := G \setminus \{u_i\}$ et blanchir les voisins de u_i .
 6. Renvoyer $\bigvee_{i=0}^d \text{DSPA}(G_i, k - 1)$.
-

Validité. On montre d'abord que les règles de réduction sont correctes (cf. le lemme 1 ci-après), ce qui valide la ligne 1. Ensuite, les lignes 3 et 4 sont trivialement correctes en observant qu'après la ligne 1, $k \geq 0$. La ligne 4 est justifiée par le lemme 2. Enfin, la ligne 6 est justifiée par le fait que dans toute solution S et pour tout sommet u_0 , soit $u_0 \in S$, soit l'un de ces voisins $u_i \in S$. Dit autrement au moins un $u_i \in S$, $i \in \{0, \dots, d\}$. Or, si $u_i \in S$, on a une solution de taille $k - 1$ pour $G_i = G \setminus \{u_i\}$ où les voisins de u_i ont été blanchis. Inversement, si G_i a une solution S_i de taille $k - 1$, alors on obtient une solution de taille k pour G en ajoutant à S_i le sommet u_i qui dominera en particulier ses voisins qui sont peut être noirs dans G .

Lemme 1 *Les règles de réduction sont correctes.*

Preuve. Il est clair que les règles de réduction préserve la planarité. Soit $(G', k') = Ri(G, k)$ la nouvelle instance après l'application d'une des règles sur l'instance (G, k) .

Pour les règles R1 et R2 on a clairement que si (G', k) est VRAI, alors (G, k) est également VRAI. L'ajout d'arêtes ou de sommets déjà dominés (blanc donc) ne peut pas invalider un ensemble dominant de G' : la solution pour G' est aussi une solution pour G .

Inversement, pour la règle R1, si (G, k) est VRAI, alors toute solution S pour G le sera aussi pour G' , et donc (G', k) est VRAI, puisque la suppression d'arêtes entre sommets blancs ne peut pas modifier l'ensemble des sommets noirs dominés par S . Pour la règle R2, si (G, k) est VRAI, alors on peut trouver une solution (de même taille) pour G qui n'utilise pas le sommet blanc v absent de G' . En effet, il suffit de prendre le sommet le sommet u qui contient le voisinage de v . Le sommet v étant déjà dominé, la nouvelle solution pour G en est une aussi pour G' , et donc (G', k) est VRAI.

Pour la règle R3, si (G, k) est VRAI, alors il existe une solution de taille k pour G qui utilise le sommet v mais pas le sommet noir u de degré 1. Dit autrement G' , le graphe

G sans v et en blanchissant son voisinage, possède une solution de taille $k - 1$. Donc, $(G', k - 1)$ est VRAI. Inversement, si $(G', k - 1)$ est VRAI, alors en ajoutant v à la solution de G' on domine tous les sommets noirs de G' ainsi que u et v . Donc, c'est une solution de taille k pour G : (G, k) est VRAI. \square

Lemme 2 *Dans une instance réduite il existe toujours parmi les sommets noirs un de degré ≤ 7 , même si on limite la règle R2 aux sommets blancs de degré ≤ 3 .*

On ne démontrera pas ce lemme, dont la preuve utilise activement la planarité. Elle est longue et fastidieuse et sort du cadre de ce cours. On remarquera que le graphe de la figure 2.7 montre que la borne sur le degré peut être atteinte.

Complexité. Le nombre de nœuds de l'arbre est au plus $\sum_{i=0}^{k-1} 8^i = O(8^k)$. La ligne la plus coûteuse est la ligne 1. Chaque règle réduit l'instance d'au moins un sommet ou une arête. Donc on applique au plus $n + m = O(n)$ règles. Chaque règle se calcule en temps $O(n)$, ce qui fait un total de $O(n^2)$ pour la ligne 1. Au total la complexité de l'algorithme est $O(8^k \cdot n^2)$.

[Exercice : Montrer que la réduction de l'instance peut être effectuée en temps $O(n)$ en modifiant éventuellement les règles.] \square

Dans les sections 2.8.4 et 2.8.5, nous verrons d'autres algorithmes pour ENSEMBLE DOMINANT pour les graphes planaires, de meilleure complexité mais pas forcément plus efficace en pratique.

[Exercice : montrer que si F est une forêt, alors $\text{DPSA}(F, k)$ a une complexité polynomiale, quelle est sa complexité? – Aide : On n'atteint jamais la ligne 4. Proposer un algorithme linéaire pour trouver un ensemble dominant de taille minimum.]

2.5.4 Améliorations : raccourcissement de l'arbre

Voici un tableau permettant de mieux comprendre les enjeux de l'explosion combinatoire pour différents algorithmes FPT en k pour COUVERTURE PAR SOMMETS. Le meilleur algorithme connu pour ce problème explore un espace de solution de taille bornée par $f(k) = 1.28^k$.

k	$f(k) = 2^k$	$f(k) = 1.49^k$	$f(k) = 1.32^k$	$f(k) = 1.28^k$
10	$\approx 10^3$	≈ 54	≈ 16	≈ 12
20	$\approx 10^6$	≈ 2909	≈ 258	≈ 140
30	$\approx \boxed{10^9}$	$\approx 10^5$	≈ 4140	≈ 1650
40	$\approx 10^{12}$	$\approx 10^7$	$\approx 10^4$	$\approx 10^4$
50	$\approx 10^{15}$	$\approx \boxed{10^9}$	$\approx 10^6$	$\approx 10^5$
75	$\approx 10^{22}$	$\approx 10^{13}$	$\approx \boxed{10^9}$	$\approx 10^8$
100	$\approx 10^{30}$	$\approx 10^{17}$	$\approx 10^{12}$	$\approx \boxed{10^{10}}$
500	$\approx 10^{150}$	$\approx 10^{85}$	$\approx 10^{60}$	$\approx 10^{53}$

On considère généralement que le terme $f(k)$ devient un frein conséquent à la complexité d'un algorithme FPT lorsque qu'il atteint l'ordre du milliard, soit 10^9 voir 10^{10} .

Théorème 7 COUVERTURE PAR SOMMETS *de taille k pour les graphes à n sommets et m arêtes peut être résolu en temps $O(5^{k/4} \cdot (n + m)) = O(1.495^k \cdot (n + m))$.*

Preuve. On note $\Delta(G)$ le degré maximum d'un sommet de G , et $N(u)$ l'ensemble des voisins du sommet u .

L'amélioration consiste à remarquer que lorsque que $\Delta(G) \leq 2$, alors une couverture par sommets de taille optimale peut être facilement calculée. En effet, si $\Delta(G) \leq 2$, alors G est une union disjointe de sommets isolés, de cycles et de chemins. Clairement, la taille d'une couverture par sommets minimum pour un chemin ou un cycle de longueur t (c'est-à-dire avec t arêtes) est $\lceil t/2 \rceil$ (voir figure 2.9). On peut donc se restreindre à un graphe G ayant au moins un sommet u de degré trois ou plus.

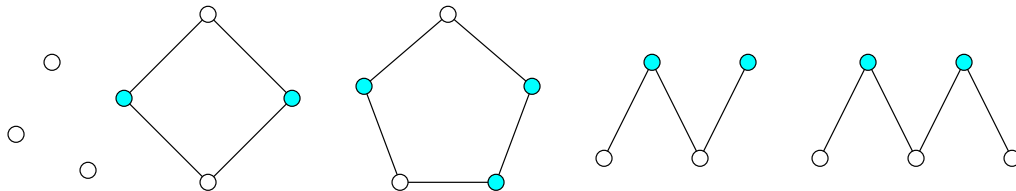


FIGURE 2.9 – Couverture optimale pour un graphe de degré maximum $\Delta \leq 2$.

On remarque aussi que pour toute couverture C et sommet u , soit $u \in C$ ou $N(u) \subseteq C$. Dans le premier cas, un sommet parmi les k recherchés peut être enlevé du graphe courant (soit u). Dans le second cas au moins trois sommets parmi les k recherchés peuvent être supprimés. Il y a donc espoir d'obtenir un arbre de recherche avec moins de sommets comme le suggère la figure 2.10.

Cela mène à l'algorithme suivant, en supposant que la fonction $\text{VCDEG2}(G, k)$ qui, étant donné que $\Delta(G) \leq 2$, retourne VRAI si et seulement si G possède une couverture par sommets de taille $\leq k$. Il est clair que la complexité de $\text{VCDEG2}(G, k)$ est en $O(n + m)$.

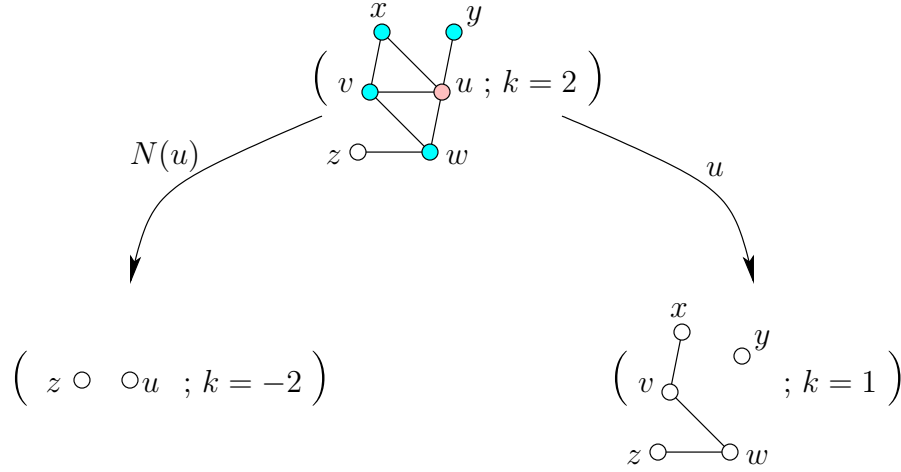


FIGURE 2.10 – Arbre de recherche pour VC2 sur la même instance que la figure 2.4.

 Algorithme VC2(G, k)

1. Si $E(G) = \emptyset$ et $k \geq 0$, renvoyer VRAI.
 2. Si $k \leq 0$, renvoyer FAUX.
 3. Si $\Delta(G) \leq 2$, renvoyer VCDEG2(G, k).
 4. Choisir un sommet u de G de degré ≥ 3 .
 5. Construire $G_1 := G \setminus \{u\}$ et $G_2 := G \setminus N(u)$.
 6. Renvoyer $\text{VC2}(G_1, k-1) \vee \text{VC2}(G_2, k - \deg_G(u))$.
-

Validité de l'algorithme : il faut, comme précédemment montrer que (G, k) est VRAI si et seulement si $(G_1, k-1)$ ou $(G_2, k - \deg_G(u))$ est VRAI. Notez que si dans le second appel de la ligne 6, k devient < 0 (si $\deg(u) > k$), alors on renverra FAUX à cause de la ligne 2, ce qui est bien correct. [Exercice : finir la preuve de la correction].

Calculons la complexité de $\text{VC2}(G, k)$. Les instructions les plus coûteuses (en dehors des appels récursifs) sont celles des lignes 3 et 5 qui prennent un temps $O(n+m)$. Soit a_k le nombre maximum de fois que l'une des instructions 3 ou 5 est exécutée lors d'un appel à $\text{VC2}(G, k)$. On a donc une complexité totale pour l'algorithme de $O(a_k \cdot (n+m))$.

On a $a_0 = 0, a_1 = 1, a_2 = 2$ (pour ce dernier cas, il faut réfléchir un peu). De manière plus générale, $a_k \leq 1 + a_{k-1} + a_{k-3}$ pour tout $k > 2$. On pose $c = 5^{1/4} \approx 1.495$ et $\alpha = \max_{k \in \{0,1,2\}} \{(a_k + 1)/c^k\} \approx 1.34$. Montrons par récurrence que $a_k \leq \alpha \cdot c^k - 1$, pour tout $k \geq 0$. Pour $k \in \{0,1,2\}$, on a, à cause du max dans la définition de α :

$$\alpha \cdot c^k - 1 \geq ((a_k + 1)/c^k) \cdot c^k - 1 = a_k .$$

Donc c'est vrai pour $k \in \{0, 1, 2\}$. Supposons la récurrence vraie jusqu'à $k - 1$, on va la prouver pour k :

$$\begin{aligned} a_k &\leq 1 + a_{k-1} + a_{k-3} = 1 + (\alpha \cdot c^{k-1} - 1) + (\alpha \cdot c^{k-3} - 1) \\ &\leq \alpha \cdot c^k \cdot (c^{-1} + c^{-3}) - 1 \\ &< \alpha \cdot c^k - 1 \end{aligned}$$

car $c^{-1} + c^{-3} = 5^{-1/4} + 5^{-3/4} \approx 0.96 < 1$. D'où au total une complexité de $O(5^{k/4} \cdot (n+m))$. \square

2.6 Réduction à un noyau : « kernalisation »

2.6.1 Principe et propriété

L'idée principale de la technique de kernalisation est de réduire une instance I d'un problème paramétré par k , à une équivalente I' de taille au plus $g(k)$, pour une certaine fonction g .

Le plus souvent g est polynomiale, voir linéaire. Ensuite I' est analysée exhaustivement, et une solution pour I' , s'il elle existe, est prolongée pour I . On dit que I' est un « noyau » (*kernel* en Anglais) pour I .

Plus formellement :

Définition 4 *Un problème Π de paramètre k possède un noyau si Π est décidable et si toute instance se réduit à une instance de taille au plus $g(k)$, pour une certaine fonction g .*

On rappelle qu'une « instance I se réduit à une instance I' » si, en temps polynomial en la taille de I , il est possible de construire I' telle que I est acceptée si et seulement si I' l'est. On a déjà vu l'utilisation de réductions dans la construction d'ensembles dominants pour les graphes planaires (cf. la preuve du théorème 6).

Par rapport aux problèmes FPT, cette technique, si elle s'applique, mène à un facteur additif exponentiel en k plutôt que multiplicatif pour la complexité en temps :

$$T(n, k) = n^{O(1)} + f(g(k)) .$$

Le point surprenant est que cette technique s'applique toujours aux problèmes FPT. En effet, on a :

Proposition 2 *Un problème Π de paramètre k possède un noyau si et seulement si Π est FPT en k .*

Preuve.

\Rightarrow Si Π possède un noyau, on peut alors résoudre l'instance I de taille n et de paramètre k en la réduisant, en temps $n^{O(1)}$, à une instance I' (équivalente) de taille $|I'| \leq g(k)$. De plus, Π est décidable. Donc on peut résoudre I' en temps $f(|I'|) = f(g(k))$ pour une certaine fonction f (au pire on teste toutes les possibilités). Au final, on a résolu I en temps $n^{O(1)} + f(g(k)) \leq f(g(k)) \cdot n^{O(1)}$, ce qui montre bien que Π est FPT en k .

\Leftarrow Soit I une instance de taille n pour Π que l'on suppose FPT en k . On peut alors résoudre I en temps $f(k) \cdot n^c$ pour une certaine constance $c > 0$. Il y a deux cas :

- Si $f(k) \geq n$, alors I est elle-même son noyau : sa taille est $|I| = n \leq f(k)$.
- Si $f(k) < n$, alors on peut résoudre I en temps $f(k) \cdot n^c < n^{c+1}$, c'est-à-dire en temps polynomial. Soit I_k^V et I_k^F les instances de Π de plus petite taille telles que I_k^V est VRAI et I_k^F est FAUX. Il est possible qu'une de ces deux instances n'existent pas. Notons que ces instances ne dépendent que de k , et donc leur taille aussi. On peut donc construire pour I , en temps polynomial, un noyau $I' \in \{I_k^V, I_k^F\}$ de taille $\leq g(k)$ choisi suivant la valeur de I qui peut être déterminée en temps polynomial.

Dans les deux cas nous avons montré que I possède un noyau, donc Π possède un noyau. \square

Autrement dit, si l'on sait que le problème est FPT en k , cela ne coûte rien de chercher d'abord un noyau pour ce problème, et bien sûr, le plus petit possible. Malheureusement, la proposition 2 ne nous dit quelles réductions sont à appliquer pour trouver le noyau.

2.6.2 Exemple de noyau

COUVERTURE DE POINTS PAR DES LIGNES :

Instance: un ensemble de points P du plan et un entier k

Paramètre: k

Question: est-ce que P peut être couvert par k droites ?

Ce problème est NP-complet (cf. [GL06]). Nous allons montrer que ce problème possède un noyau. On le montre en appliquant, autant que possible, la réduction suivante : s'il existe une droite couvrant un ensemble S de plus de k points, alors on les supprime de P (cf. figure 2.11). L'instance (P, k) se réduit alors en $(P \setminus S, k - 1)$.

Nous avons trois points à démontrer :

1. Il s'agit bien d'une réduction, valide donc. En effet, si l'instance $(P \setminus S, k - 1)$ est acceptée, alors (P, k) a aussi une solution. Inversement, si (P, k) a une solution alors $(P \setminus S, k - 1)$ doit aussi en avoir une, car nécessairement une droite suffit à couvrir S dans la solution de (P, k) , et donc les $k - 1$ autres droites suffisent à couvrir $P \setminus S$. En effet, si une seule droite suffisait pas à couvrir S , alors $|S|$ droites différentes devraient être utilisées pour couvrir S , ce qui est impossible puisque $|S| > k$.

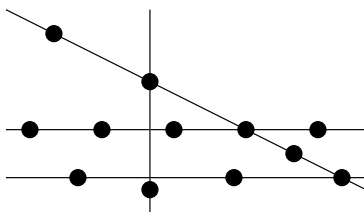


FIGURE 2.11 – Couverture de points par des lignes de taille optimale [Exercice : pourquoi est-elle optimale ?].

2. La réduction est polynomiale. En effet, on peut supposer que chaque droite couvre au moins deux points ou alors est horizontale, si bien qu'il y a au plus $n(n-1)/2 + n = O(n^2)$ droites candidates à tester. La réduction est donc polynomiale.
3. Le noyau (c'est-à-dire l'instance réduite finale) est bien de taille au plus $g(k) \leq k^2$. Si on ne peut plus appliquer la réduction et qu'il existe plus de k^2 points, alors il n'y a pas de solution.

On a donc montré que ce problème a un noyau de taille k^2 .

Remarque : On peut démontrer (cf. [Nie06, pp. 74-80]) qu'ENSEMBLE DOMINANT de taille k pour les graphes planaires possède un noyau linéaire en k calculable en temps $O(n^3)$, ce qui donc peut être combiné avec l'algorithme vu au paragraphe 2.5.3. Le noyau de plus petite taille connu pour ce problème possède $67k$ sommets, la taille de tout noyau étant au moins $2k$ [CFKX05]. En fait tout problème FPT exprimable dans une certaine logique et vérifiant une certaine condition de « compacité » (ENSEMBLE DOMINANT fait partie de ces problèmes) possède un noyau de taille linéaire si l'on restreint aux graphes de genre borné [BFL⁺09].

2.6.3 COUVERTURE PAR SOMMETS

Théorème 8 COUVERTURE PAR SOMMETS *de taille k pour les graphes à n sommets peut être résolu en temps $O(kn + 5^{k/4} \cdot k^2)$.*

Preuve. On va montrer que le problème COUVERTURE PAR SOMMETS de taille k possède un noyau de taille $O(k^2)$, calculable en temps $O(kn)$. La complexité découle de l'algorithme VC2 appliqué au noyau (théorème 7).

Observation : si un graphe a un sommet de degré $> k$ alors il doit appartenir à toute couverture de taille k sur ce graphe, puisque toute couverture doit contenir soit u soit $N(u)$ et ce pour tout sommet u . Or le second cas est exclu car $|N(u)| > k$.

Soit G un graphe à n sommets et m arêtes. On considère l'algorithme suivant, basé sur l'observation précédente :

Algorithme VC3(G, k)

1. Si $E(G) = \emptyset$ et $k \geq 0$, renvoyer VRAI.
2. Si $|E(G)| > k \cdot \Delta(G)$, renvoyer FAUX.
3. On pose $H := G$ et $p := 0$.
Tant que $p < k$ et $\exists u \in V(H)$ tels que $\deg_H(u) > k - p$ faire :
– $H := H \setminus \{u\}$ et $p := p + 1$.
4. Si $|E(H)| > (k - p)^2$, renvoyer FAUX.
5. Supprimer les sommets isolés de H .
6. Renvoyer VC2($H, k - p$).

Montrons que l'algorithme ci-dessus est correct. Le test de l'étape 2 est valide, car un sommet u appartenant à une couverture ne peut couvrir qu'au plus $\deg(u) \leq \Delta(G)$ arêtes.

Les p sommets de l'étape 3 appartiennent nécessairement à la solution recherchée. En effet, d'après l'observation un sommet de degré $> k$ doit appartenir à la solution, et plus généralement, après i itérations de la boucle « tant que » tout sommet de degré $> k - i$ doit être dans une couverture de taille $k - i$ (i sommets ayant été déjà placés dans la couverture).

Après l'étape 3, G possède une couverture de taille k si et seulement si H possède une couverture de taille $k - p \geq 0$. Le test de l'étape 4 se justifie de la même manière que celui de l'étape 2. Si $|E(H)| > (k - p) \cdot \Delta(H)$, alors la couverture n'existe pas. Or, à cause de la double condition du « tant que » de l'étape 3, soit $p = k$, soit $\Delta(H) \leq k - p$. Dans les deux cas le test est valide.

On est donc ramené à déterminer si H possède une couverture de taille $k - p$, ce qui justifie les deux dernières étapes.

Calculons la complexité de cet algorithme. On note m le nombre d'arêtes de G . L'étape 1 s'implémente en $O(kn)$ de la manière suivante. On compte les arêtes de G tout en veillant à ne pas dépasser $k(n - 1)$. Si $m > k(n - 1)$ il faut bien sûr renvoyer FAUX. Dans l'autre cas, $m \leq k(n - 1)$. On a alors déjà calculé $\Delta(G)$ en temps $O(kn)$ et on peut faire le test plus fin de l'étape 1. De la même façon, l'étape 4 prend un temps $O(k^2) = O(kn)$ et l'étape 5, $O(n)$. Donc les étapes 1, 4 et 5 prennent un temps $O(kn)$.

C'est un peu plus compliqué pour implémenter efficacement l'étape 3. Notons qu'après l'étape 1, on a $m \leq k(n - 1)$. On peut par exemple, utiliser une structure de données (construite juste avant l'étape 3) permettant de maintenir une table des degrés des sommets (on peut coder par 0 le fait que le sommet est supprimé). Cela peut être initialisé en temps $O(n + m)$ en un simple parcours de G (représenté par défaut par sa liste d'adjacence). La mise à jour de cette structure se fait en temps $O(\deg(u)) = O(n)$ (il faut faire attention de ne supprimer les sommets qu'une fois!). Trouver un sommet de degré $\geq k - p$ se fait aussi en temps $O(n)$. Donc, au total l'étape 3 prend un temps $O(pn) = O(kn)$.

Pour préparer l'appel à VC2 à l'étape 6, on repasse à une représentation par listes d'adjacence pour H en rétablissant les listes réelles de voisins. Cela se fait en parcourant

G , en temps $O(n + m) = O(kn)$, et en ne gardant que les sommets de degré > 0 , ce qui à la volée implémente l'étape 5.

Après l'étape 4, H possède au plus $(k - p)^2$ arêtes. Donc après l'étape 6, il possède au plus $2(k - p)^2$ sommets, une arête étant incidente à au plus deux sommets distincts. Le problème possède donc un noyau de taille $O(k^2)$. Notez qu'en ajoutant la règle (juste avant la ligne 5) : si $\deg(u) = 1$, alors enlever $N(u)$ et diminuer k de 1, on aurait pu montrer que le noyau possède au plus $(k - p)^2$ sommets puisque dans ce cas $|V(H)| \leq |E(H)|$.

Le temps pour l'étape 6 est de $O(5^{(k-p)/4} \cdot (k - p)^2) = O(5^{k/4} \cdot k^2)$ d'après le résultat précédent (théorème 7). La complexité totale de l'algorithme est donc : $O(kn + 5^{k/4}k^2)$. \square

Le meilleur algorithme connu a une complexité⁴ de $O(1.2738^k + kn)$ [CKX06]. Il utilise, en plus de la technique d'arbre de recherche bornée, la technique de réduction par « couronnes » (*crown*) qui donne un noyau de taille linéaire. Cela peut être vue comme une généralisation du traitement spécial des sommets de degré 1. L'idée est de déterminer un ensemble de sommets I de sorte qu'il existe une couverture par sommets optimale qui contienne tous les sommets de $N(I)$ et aucun de I (voir la figure 2.12). C'est bien le cas si $I = \{u\}$ est un sommet de degré 1.

On rappelle qu'un *couplage* est un ensemble d'arêtes indépendantes (voir le paragraphe 3.2.1 pour plus de détails).

Définition 5 Une couronne est un ensemble I non vide de sommets indépendants tel qu'il existe un couplage entre I et $N(I)$ couvrant tous les sommets de $N(I)$. La taille d'une couronne est $|N(I)|$.

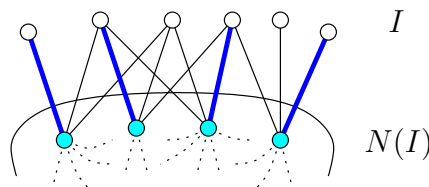


FIGURE 2.12 – Une couronne de taille 4.

Il est facile de voir que G possède une couverture par sommets de taille k si et seulement s'il en possède une de taille k contenant $N(I)$ et aucun sommet de I . Donc on peut réduire l'instance en supprimant $N(I)$ et I , et en diminuant k de $|N(I)|$.

On va démontrer l'existence d'un noyau de taille linéaire grâce au lemme suivant :

Lemme 3 *Tout graphe sans sommet isolé possédant une couverture par sommets de taille $\leq k$ qui a $> 3k$ sommets contient une couronne qui peut être calculée en temps polynomial.*

4. La complexité précise est $O(c^k \cdot k + kn)$ où $c \approx 1.2737\dots$. Et bien sûr, pour k assez grand $c^k \cdot k < 1.2738^k$.

Par ce biais on obtient un noyau d'au plus $3k$ sommets en supprimant successivement les couronnes de G . Trouver une couronne de la plus grande taille possible est polynomial [AKFLS07].

Preuve. Soit G un graphe satisfaisant l'énoncé du lemme, et soit M un couplage maximal de G , calculé par un algorithme glouton (en temps linéaire). On note X l'ensemble des sommets induit par M et I les sommets de G qui ne sont pas dans X . Bien sûr I est un ensemble indépendant. De plus $|X| = 2|M| \leq 2k$, puisque G possède une couverture par sommets de taille k . Il suit que $|I| > k$, sinon G aurait $\leq 3k$ sommets.

On construit le graphe G' induit par les arêtes entre X et I . Il s'agit du graphe G dans lequel les arêtes entre les sommets de X ont été supprimées. Ce graphe est biparti. Comme on le verra plus tard dans le cours (cf. le théorème 16), G' , et plus généralement tout graphe biparti, possède une couverture par sommets C dont la taille (minimum) égale la taille (maximum) d'un couplage M' de G' . Cette couverture (ou ce couplage) peut être construite en temps polynomial. Chaque arête de M' contient exactement un sommet de C qui est soit dans X soit dans I .

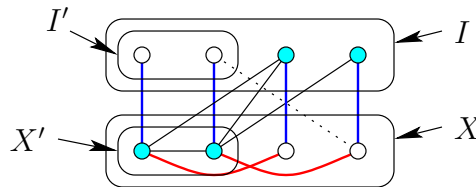


FIGURE 2.13 – Couplage maximal de G (en rouge), couplage maximum de G' (en bleu), couverture minimum (en cyan), et couronne I' avec $N(I') = X'$. L'arête en pointillée ne peut exister.

La couverture C contient au moins un sommet de X . Sinon, M' aurait une taille $= |I| > k$ (I ne possède pas de sommet isolé). Cela n'est pas possible car M' est aussi un couplage de G , et G a une couverture par sommets de taille k seulement. Soit $X' = X \cap C$, et soit I' l'ensemble des sommets induits par les arêtes de M' dont une extrémité est dans X' (cf. figure 2.13). On remarque qu'il ne peut exister d'arête entre un sommet de I' et de $X \setminus X'$, puisqu'aucune de ces extrémités n'est dans C . C'est aussi vrai dans G . Autrement dit, $N(I') = X'$. L'ensemble I' forme donc une couronne dans G . \square

Cette réduction à un noyau de taille linéaire fait appel à l'algorithme de couplage maximum dans un graphe biparti qui peut être calculé en temps $O(m\sqrt{n}) = O(kn^{3/2})$ à l'aide d'une technique de flôts (voir le paragraphe 3.2.1 pour plus de détails). Dans la pratique, il faut donc calculer d'abord en temps $O(kn)$ un noyau de $O(k^2)$ sommets et arêtes, puis appliquer la réduction au noyau de taille linéaire en temps $O(k^3)$ avant d'appliquer la recherche exponentielle en c^k . Cela donne une complexité totale en $O(kn + k^3 + 5^{k/4} \cdot k) = O(kn + 5^{k/4} \cdot k)$.

2.6.4 Noyau et approximation

De nombreuses recherches ont pour objectif de construire, en temps polynomial, des noyaux de taille linéaire en le paramètre k . La motivation n'est pas seulement sur le gain en temps, qui dans certain cas n'est pas significatif, mais pour une raison plus profonde : les algorithmes d'approximation.

Supposons par exemple que pour COUVERTURE PAR SOMMETS de taille k il existe un noyau de ck sommets pour une certaine constante $c \geq 1$. Alors cette réduction peut répondre (en temps polynomial) au problème suivant :

COUVERTURE PAR SOMMETS c -APPROXIMÉE :

Instance: un graphe G et un entier k

Question: est-ce que G possède une couverture par sommets de taille k ?

Si oui en renvoyer une de taille $\leq ck$, sinon répondre non.

En effet, si lors de la réduction un rejet se produit, alors G ne possède pas de couverture par sommets de taille k , et on peut répondre non. En revanche si la réduction aboutit, on peut fournir une couverture pour G composée de tous les sommets de la réduction (et de ceux sélectionnés lors de la réduction), ce qui fait au plus ck sommets. On en déduit alors une c -approximation pour le calcul de la taille optimale : on cherche (par dichotomie) le plus petit k qui donne une réponse positive.

Pour COUVERTURE PAR SOMMETS, il existe une méthode plus sophistiquée que celle du lemme 3 donnant un noyau de taille $2k$ (voir [Khu02]). Comme on le verra pas la suite, le meilleur algorithme connu pour COUVERTURE PAR SOMMETS donne un facteur d'approximation de $2 - o(1)$. Pour cette raison, les gens pensent qu'il n'existe pas de noyau avec moins de $(2 - \epsilon)k$ pour COUVERTURE PAR SOMMETS, à moins bien sûr que $P=NP$. Mais la borne de $(2 - \epsilon)k$ sur la taille minimal d'un noyau n'est pas démontrée. La meilleure borne inférieure connue sur le noyau de COUVERTURE PAR SOMMETS est de $1.36k$ sommets, sauf si $P=NP$. Ce résultat résulte d'un théorème profond de la théorie de l'approximation appelé « Théorème PCP » qui ne sera pas abordé dans ce cours.

Pour terminer cette section, montrons un autre exemple (certes artificiel) de réduction à un noyau de taille linéaire.

Proposition 3 ENSEMBLE INDÉPENDANT de taille k pour les graphes planaires a un noyau planaire de moins de $4k$ sommets.

Preuve. On considère G un graphe planaire à n sommets. En temps polynomial, il est possible de donner pour G une 4-coloration. Notons que, par définition de la coloration, les sommets d'une même couleur forme un ensemble indépendant pour G . Soit C l'ensemble des sommets de la couleur la plus fréquente. Évidemment, $|C| \geq n/4$ et C est un

ensemble indépendant. Si $k \leq n/4$, alors répondre OUI, puisque si G possède un ensemble indépendant de taille $|C|$ il en existe un de taille k (prendre n'importe quel sous-ensemble de C de taille k). Sinon, c'est que $k > n/4$ et donc $n < 4k$. Autrement dit G lui-même est un noyau de $< 4k$ sommets. \square

Pour compléter le tableau, il est aussi connu que tout noyau pour le problème ENSEMBLE INDÉPENDANT de taille k dans les graphes planaires possède au moins $(4/3 - \epsilon)k$ sommets, pour tout $\epsilon > 0$, sauf si $P=NP$.

[Exercice : on a vu dans la remarque de la section 2.6.2, qu'ENSEMBLE DOMINANT de taille k dans les graphes planaires admet un noyau de taille linéaire, disons avec ck sommet. Construire un algorithme d'approximation de ratio $c/2$ pour ce même problème.]

2.7 Théorie des mineurs de graphes

Une possibilité pour montrer qu'un problème est FPT est d'utiliser la théorie des mineurs de graphes. Cette théorie a été activement développée par Robertson et Seymour dans le milieu des années 1980. Cette méthode doit être utilisée surtout pour la classification des problèmes (FPT ou pas?). Elle ne fournit pas vraiment d'algorithmes FPT « pratique », elle dit simplement qu'il *existe* un algorithme.

2.7.1 Définitions

On rappelle que la contraction d'une arête entre u et v consiste à (voir figure 2.14) :

1. supprimer l'arête entre u et v ,
2. identifier les sommets u et v ,
3. enlever les arêtes doubles (s'il y en a).

Le nouveau sommet a donc pour degré $\deg(u) + \deg(v) - 2 - c$ où c est le nombre de voisins communs à u et v (autres que u, v).

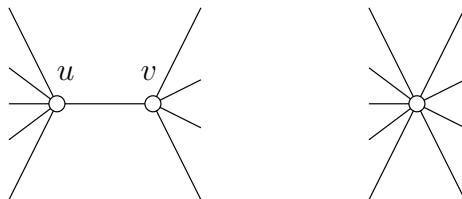


FIGURE 2.14 – Contraction de l'arête $\{u, v\}$.

Définition 6 (Mineur) *Un mineur d'un graphe G est un sous-graphe d'un graphe obtenu à partir de G en contractant des arêtes.*

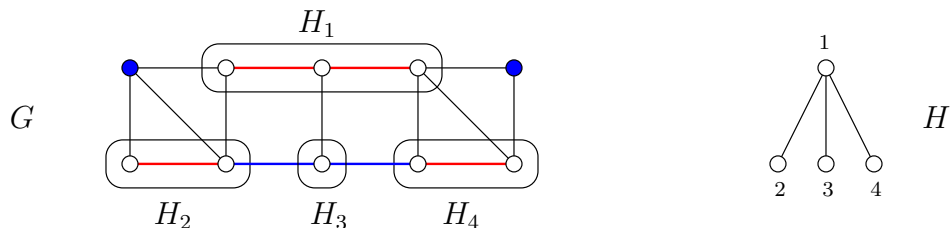


FIGURE 2.15 – Un mineur H d'une grille G . Les contractions sont en rouges, les suppressions en bleues. (H_1, \dots, H_4) est un model de H dans G . Plusieurs models sont possibles.

Si H est un mineur de G , alors il n'est pas difficile de voir que H peut être obtenu à partir de G en appliquant les trois opérations suivantes, l'ordre n'a pas d'importance :

- suppression d'arêtes.
- contraction d'arêtes.
- suppression de sommet isolé.

Une autre façon de montrer que H est un mineur de G est, en supposant que $V(H) = \{1, \dots, h\}$, de construire ses sous-graphes connexes et disjoints H_1, \dots, H_h de G tels que s'il existe une arête dans H entre i et j , alors il existe une arête de G entre H_i et H_j . La suite (H_1, \dots, H_h) est parfois appelée *model* de H dans G (voir la figure 2.15).

Une famille de graphes \mathcal{F} est dite *close par mineurs* si tous les mineurs des graphes de \mathcal{F} sont aussi dans \mathcal{F} . Les forêts sont, par exemple, une famille close par mineurs. Les graphes planaires aussi. Les graphes de degré borné ne le sont pas (figure 2.16), tout comme les familles de graphes qui ne sont pas close par sous-graphes, comme les graphes de diamètre au plus k par exemple.

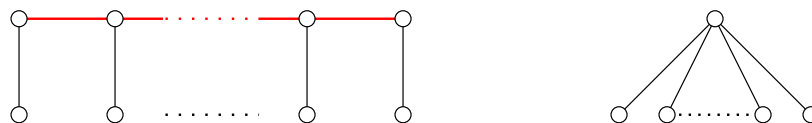


FIGURE 2.16 – La famille des graphes de degré borné n'est pas close par mineurs.

2.7.2 Théorème des mineurs de graphes

Théorème 9 (Graph Minor Theorem – [RS04]) *Soit \mathcal{F} une famille de graphes close par mineurs. Alors il existe un ensemble \mathcal{O} fini de graphes tels que $G \in \mathcal{F}$ si et seulement si G ne contient aucun mineur appartenant à \mathcal{O} .*

L'ensemble \mathcal{O} est parfois appelé « obstructions » de \mathcal{F} , ou encore les mineurs exclus de \mathcal{F} . Bien sûr \mathcal{O} ne dépend que de \mathcal{F} . La difficulté⁵ dans la preuve du théorème 9 est

5. La preuve qui s'étale sur plus de 20 ans (de 1983 à 2004) contient plus de 500 pages.

de montrer qu'il existe un ensemble \mathcal{O} fini, puisque sinon il suffit de prendre l'ensemble de tous les graphes qui ne sont pas dans \mathcal{F} . En fait, le théorème 9 est équivalent à dire que dans toute famille infinie de graphes, il existe toujours deux graphes tels que l'un est mineur de l'autre.

Pour les graphes planaires, $\mathcal{O} = \{K_5, K_{3,3}\}$, pour les graphes planaires-extérieurs $\mathcal{O} = \{K_4, K_{2,3}\}$, pour les forêts $\mathcal{O} = \{K_3\}$, pour les graphes série-parallèles $\mathcal{O} = \{K_4\}$.

Théorème 10 ([RS95]) *Pour tout graphe H , il existe une constante $c(H)$ telle qu'on peut décider si un graphe G à n sommets possède H comme mineur en temps au plus $c(H) \cdot n^3$.*

L'algorithme cubic n'est pas du tout efficace en pratique car la constante $c(H)$, résultante de la preuve, est gigantesque. En effet, on a évalué que

$$c(H) > \text{tour}(2, 3 + \text{tour}(2, \text{tour}(2, \lfloor |V(H)|/2 \rfloor))) \quad \text{où} \quad \text{tour}(k, h) = k^{\left. \begin{matrix} k \\ \vdots \\ k \end{matrix} \right\} h}$$

est une tour d'exponentielles en k de hauteur h . Plus formellement : $\text{tour}(k, 0) = k^0 = 1$ et $\text{tour}(k, h) = k^{\text{tour}(k, h-1)}$. Pour $|V(H)| = 2$ cela fait déjà $\text{tour}(2, 7) = 2^{2^{65536}}$. Rappelons qu'on estime à 2^{400} le nombre de particules dans l'Univers ... Il est donc plus utile de développer des algorithmes spécifiques que d'appliquer le théorème 10.

Pour les familles citées ci-dessus (graphes planaires, planaire-extérieures, série-parallèles, *etc.*) il existe des algorithmes linéaires. De même, savoir si un graphe possède K_5 comme mineur peut être déterminé en temps linéaire.

Bien sûr la constante $c(H)$ est liée à la preuve du théorème 10, et il est concevable qu'une autre preuve pourrait aboutir à une constante plus raisonnable, même si $c(H)$ est nécessairement exponentielle en la taille de H .

2.7.3 Applications

Un graphe est *sans entrelacement* (*linkless* en Anglais) s'il est possible de le dessiner dans \mathbb{R}^3 sans croisement d'arêtes⁶ tel qu'il n'existe pas deux cycles disjoints entrelacés⁷. Deux courbes de \mathbb{R}^3 sont entrelacées si leur projection sur \mathbb{R}^2 s'intersectent (voir figure 2.17).

Au problème de savoir si un graphe est sans entrelacement, on ne connaissait aucun algorithme. Jusqu'en 1995, il était ouvert de savoir si ce problème était ne serait-ce que décidable. En effet, comment tester tous les plongements possibles de \mathbb{R}^3 ? Et comment tester qu'un plongement donné est sans entrelacement?

6. On parle aussi de plongement dans \mathbb{R}^3 . C'est toujours possible quelque soit le graphe.

7. Ceci n'est pas possible pour K_6 .

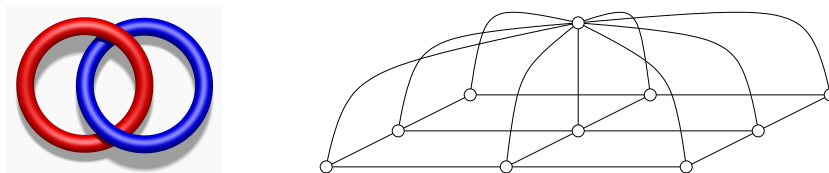


FIGURE 2.17 – Courbes entrelacées et plongements sans entrelacement. Tout graphe planaire augmenté d'un sommet possède un plongement sans entrelacement.

En fait, un graphe est sans entrelacement⁸ si et seulement si il ne contient aucun des sept graphes de la figure 2.18 comme mineurs [RST95]. Le théorème 10 donne alors un algorithme en $O(n^3)$. Récemment, un algorithme en $O(n^2)$ a été présenté et qui dans le cas positif donne un plongement sans entrelacement [KKM09]. Cet algorithme n'utilise pas la détection de mineurs interdits.

Notons qu'on ne connaît toujours pas d'algorithme permettant de vérifier si un plongement dans \mathbb{R}^3 est sans entrelacement.

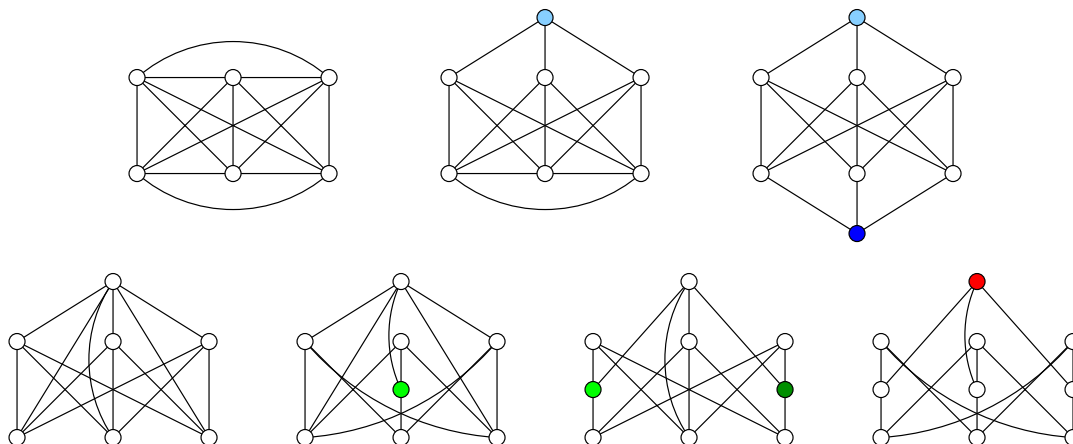


FIGURE 2.18 – Les mineurs exclus pour les graphes sans entrelacement [A REFAIRE].

En fait ces sept graphes, dont celui de Petersen, sont tous équivalents à K_6 par échanges $Y-\Delta$ (ou l'opération inverse $\Delta-Y$) : on remplace un sommet ayant au moins trois voisins par un triangle connectant les trois sommets. La figure 2.19 montre une autre relation inattendue entre K_6 et le graphe de Petersen.

Un problème similaire est de savoir si un graphe G possède un plongement dans \mathbb{R}^3 tel que tout cycle est *sans-nœud* (*knotless* en Anglais). La figure 2.20 montre une courbe qui

8. Notons que cette famille est close par mineurs : la contraction d'une arête ne peut pas entrelacer deux cycles qui ne l'étaient pas au départ. En fait, peu importe l'explication, car toute famille caractérisée par une liste de mineurs exclus est forcément close par mineur.

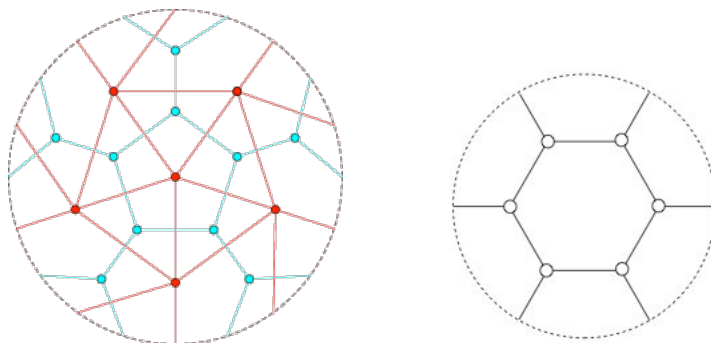


FIGURE 2.19 – Le graphe de Petersen et K_6 sont duals sur le plan projectif; $K_{3,3}$ sur le plan projectif.

contient un nœud. Le graphe K_7 n'a pas de plongement sans-nœud. La famille des graphes ayant un plongement sans-nœud est close par mineurs, et donc possède un algorithme de reconnaissance en $O(n^3)$.

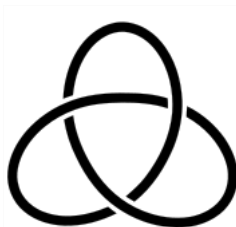


FIGURE 2.20 – Courbe avec un nœud.

De manière plus générale, en combinant les théorèmes 9 et 10 on peut savoir si un graphe G est dans une famille \mathcal{F} fixée close par mineurs en temps $c \cdot n^3$, où c ne dépend que de \mathcal{F} .

Prenons un exemple. Soit \mathcal{F}_k l'ensemble des graphes ayant un COUVERTURE PAR SOMMETS de taille $\leq k$. Résoudre COUVERTURE PAR SOMMETS de taille $\leq k$ est, par définition, équivalent à savoir si un graphe donné appartient à \mathcal{F}_k . Or, la famille \mathcal{F}_k est close par mineurs. Donc, par le théorème 9, il existe un ensemble fini \mathcal{O}_k de mineurs exclus caractérisant \mathcal{F}_k . On a donc l'existence d'un algorithme qui en temps $O(|\mathcal{O}_k| \cdot n^3)$ détermine si $G \in \mathcal{F}_k$ ou pas.

Un autre exemple est FEEDBACK VERTEX-SET de taille k . Il s'agit de déterminer si un graphe G possède un ensemble de sommets S de taille k tel que $G \setminus S$ est acyclique, c'est-à-dire une forêt. Ce problème est NP-complet, et la famille de graphes associés est close par mineurs⁹. Ce problème est donc FPT en k .

9. La contraction d'arête ne peut faire apparaître de cycle.

Un autre exemple est PLANAR FACE COVER de taille k . Il s'agit de déterminer si un graphe planaire G possède dessin où les bords de k faces couvrent tous les sommets de G . Ce problème est NP-complet, et la famille de graphes associés est close par mineurs¹⁰. Ce problème est donc FPT en k .

Malheureusement, cette théorie ne peut être appliquée aux problèmes ENSEMBLE DOMINANT et ENSEMBLE INDÉPENDANT de taille k . Pour ENSEMBLE DOMINANT ce n'est déjà pas clos par sous-graphes, et pour ENSEMBLE INDÉPENDANT ce n'est pas clos par contraction d'arêtes (cf. figure 2.21). Notons aussi que cette technique produit des algorithmes non-uniformes, c'est-à-dire spécifique pour chaque valeur du paramètre.



FIGURE 2.21 – La suppression d'une arête du graphe fait augmenter la taille du plus petit l'ensemble dominant (en cyan). La contraction d'une arête fait diminuer la taille du plus grand ensemble indépendant (en vert).

Cette méthode n'est pas très pratique, car, bien que finies, les obstructions peuvent être très nombreuses et de grande taille. La fonction $f(k)$ (dans l'expression de la complexité FPT) croît très rapidement avec k . Pour fixer les idées, le nombre d'obstructions pour les graphes toriques (dessinables sur un tore sans croisement d'arêtes) n'est pas connu : c'est au plus 16 629 (en 2002, cf. [GMC09]). De plus, on peut démontrer qu'il n'existe pas d'algorithme pour calculer l'ensemble d'obstructions d'une famille arbitraire \mathcal{F} close par mineurs, si la famille \mathcal{F} est représentée par une machine de Turing qui énumère ses éléments [FL89]. Il est donc de loin préférable d'utiliser des algorithmes spécifiques pour savoir si un graphe est planaire ou torique (il y en a des linéaires).

2.8 Réduction aux graphes de *treewidth* bornée

L'idée est de paramétrer le problème par la largeur arborescente (*treewidth* en Anglais) du graphe, de calculer une décomposition arborescente et d'utiliser la programmation dynamique.

Avant de donner un sens précise à « largeur arborescente » et « décomposition arborescente », observons que beaucoup de problèmes NP-complets deviennent polynomiaux si le graphe est un arbre (ou une forêt). Citons par exemple COUVERTURE PAR SOMMETS qui se résout trivialement en temps linéaire comme ceci (la preuve de l'optimalité est laissée en exercice) :

Algorithmme VC-FORÊT(T, k)

10. Une solution reste stable par la contraction d'une arête.

1. Si $E(T) = \emptyset$, renvoyer VRAI.
2. Si $k = 0$, renvoyer FAUX.
3. Choisir une arête $\{u, v\}$ de T telle que u soit une feuille.
4. Renvoyer VC-FORÊT($T \setminus \{v\}$, $k - 1$).

Cet algorithme travaille progressivement à partir des feuilles (et de leurs parents) en remontant vers la racine. Dans ce processus, il n'y a jamais de remise en question des décisions prises (*back tracking*), ce qui explique le temps polynomial. De manière très intuitive on peut travailler de manière indépendante sur chaque partie de l'arbre (au départ les feuilles), ce qui permet d'avancer très rapidement. Seule se pose la question de la fusion des solutions partielles.

L'espoir consiste donc à se dire que ce genre d'algorithme continuera de fonctionner si le graphe d'entrée est suffisamment « proche » d'un arbre. On va donc chercher à décomposer le graphe pour produire une « arborescente ».

2.8.1 Décomposition arborescente

Définition 7 Une décomposition arborescente d'un graphe G est un arbre T dont les nœuds, appelés sacs, sont des sous-ensembles de sommets de G tels que :

1. $\bigcup_{X \in V(T)} = V(G)$;
2. $\forall \{u, v\} \in E(G), \exists X \in V(T), u, v \in X$; et
3. $\forall u \in V(G)$, l'ensemble des sacs contenant u induit un sous-arbre de T .

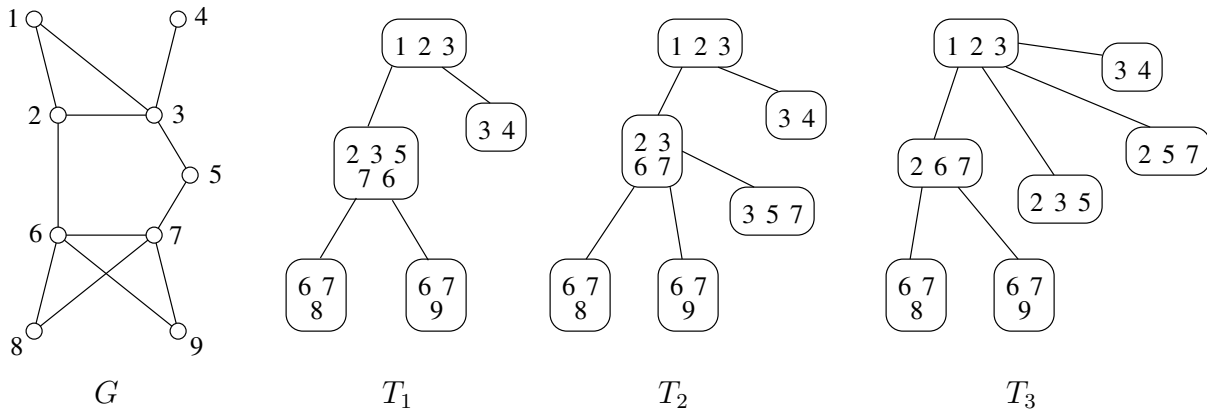


FIGURE 2.22 – Exemples de décompositions arborescentes d'un graphe G .

Une propriété fondamentale des décompositions arborescentes est la suivante. Soient $e = \{X, Y\}$ une arête de T , et soient A, B les deux sous-arbres de $T \setminus \{e\}$ (disons que A est la composante contenant X). Alors, si dans G on supprime les sommets de $X \cap Y$,

on déconnecte le sous-graphe induit par les sommets de A de celui induit par B (voir figure 2.23). C'est due au fait que les sacs contenant un sommet, une arête, un chemin et plus généralement un sous-graphe connexe de G induisent un sous-graphe connexe de T .

En effet, comme l'ensemble X^u des sacs contenant u forme un sous-arbre T^u (3e propriété), et comme une toute arête $\{u, v\}$ de G doit appartenir dans un sac X^{uv} (2e propriété), on a que $\{u, v\} \subset X^{uv} \in T^u \cap T^v$. Or l'intersection de deux arbres est connexe. Et ceci est resté vrai pour un ensemble connexe d'arêtes.

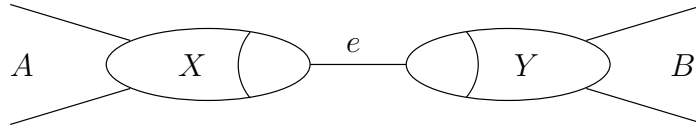


FIGURE 2.23 – Les sommets de $X \cap Y$ séparent A de B .

Autrement dit, tout chemin d'un sommet contenu dans les sacs de A vers un sommet des sacs de B passe nécessairement par un sommet de $X \cap Y$. En quelque sorte $X \cap Y$ sépare A de B dans G . C'est évidemment la même chose pour l'arbre T à savoir : tout chemin d'un nœud de A vers un nœud de B passe par l'arête e . Donc la structure du graphe se décrit plus ou moins finement par celle de l'arbre.

Bien sûr il y a plusieurs décomposition arborescente possible pour un graphe. Il y en a toujours une triviale composée d'un arbre à un seul nœud. Ce n'est pas bien sûr la plus intéressante des décompositions. Nous allons nous intéresser aux décompositions où les sacs sont les plus petits possibles.

On définit la largeur d'une décomposition arborescente T comme la valeur :

$$\text{width}(T) := \max_{X \in \mathcal{V}(T)} |X| - 1 .$$

Nous sommes intéressé par les décompositions arborescentes de largeur minimum. On définit ainsi la *largeur arborescente* d'un graphe comme étant la valeur :

$$\text{tw}(G) := \min_T \text{width}(T) .$$

La raison du « -1 » dans la définition de la largeur est de pouvoir dire que la largeur arborescente des arbres est 1. Cela serait 2 sinon. La figure 2.24 donne un exemple de décomposition arborescente d'un arbre. Une telle décomposition peut se définir récursivement en enlevant, à partir des feuilles, les arêtes de l'arbres.

Largeur arborescente de quelques graphes :

- arbre et forêt : 1
- clique à n sommets : $n - 1$
- graphe planaire-extérieur, graphe série-parallèle : 2
- graphe planaire à n sommets : $\leq 2\sqrt{6n}$

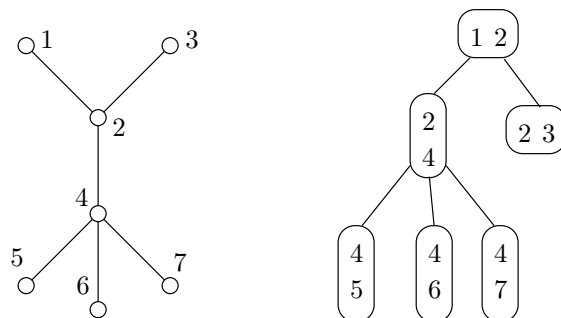


FIGURE 2.24 – Décomposition arborescente de largeur 1 pour un arbre.

– graphe qui exclu un mineur planaire H : $\leq c(H)$ (cf. l'équation 2.1 section 2.8.5)

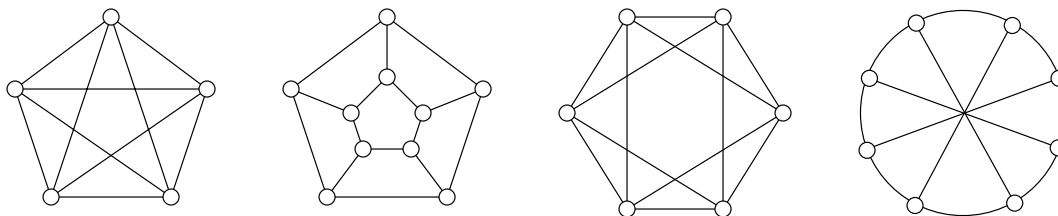
Le problème TREE-WIDTH est défini comme suit et est malheureusement NP-complet [ACP87] :

TREE-WIDTH :

Instance: un graphe G et un entier k

Question: $\text{tw}(G) \leq k$?

Cependant il est FPT en la largeur arborescente. C'est, d'une certaine façon, évident puisque la famille des graphes ayant une décomposition arborescente de largeur k est close par mineurs (cf. section 2.7). On le vérifie facilement d'après la définition 7. Malheureusement, on ne connaît pas la liste des mineurs à exclure pour chaque k . Pour $k = 1, 2$ c'est respectivement K_3 et K_4 . Pour $k = 3$, il y a K_5 , mais pas seulement. La liste complète, due à [APC90], est représentée sur la figure 2.25. pour $k \geq 4$, la liste n'est pas connue mais des algorithmes la calculant existent [AL91]. En fait, il est montré que le nombre de sommets des mineurs interdits est borné par $2^{O(k^3)}$.

FIGURE 2.25 – Les mineurs exclus pour les graphes de largeur arborescente ≤ 3 . En fait, il existe un algorithme linéaire pour reconnaître ces graphes [AP87].

En fait, il existe un algorithme en temps $2^{O(\text{tw}(G)^3)} \cdot n$ donnant une décomposition arborescente de largeur optimale : c'est donc un algorithme linéaire pour une largeur arborescente fixée. Comme on va le voir dans la suite, une décomposition arborescente de

largeur $O(\text{tw}(G))$ est souvent suffisante, et de telles décompositions peuvent être calculées en temps certes super-linéaire, mais surtout simplement exponentiel en $\text{tw}(G)$ et non pas exponentiel en $\text{tw}(G)^3$.

Pour être plus précis, en temps $2^{O(k)} \cdot n \log n$, soit l'algorithme répond que $\text{tw}(G) > k$, ou bien renvoie une décomposition arborescente pour G de largeur $O(k)$. Il en va de même pour tous les résultats présentés dans le théorème ci-dessous. Notons qu'en testant $O(\log \text{tw}(G))$ fois cet algorithme, on en déduit une décomposition de taille $O(\text{tw}(G))$. Le temps total est alors en $2^{O(\text{tw}(G))} \cdot \log \text{tw}(G) \cdot n \log n$ ce qui est aussi en $2^{O(\text{tw}(G))} \cdot n \log n$.

Évidemment, les constantes cachées dans les notations « O » jouent un rôle essentielle en pratique. Les résultats suivants (qui sont complexes et que nous admettrons dans ce cours) sont classés selon la largeur des décompositions produites (on ne retiendra que les résultats 1 et 5) :

Théorème 11 *Pour tout graphe G à n sommets et entier k , il est possible de déterminer si $\text{tw}(G) > k$, et si $\text{tw}(G) \leq k$ de construire une décomposition arborescente de largeur $\leq \omega(k)$ en temps $f(k) \cdot \text{poly}(n)$ où les paramètres sont les suivants :*

$\omega(k)$	$f(k)$	$\text{poly}(n)$	références
k	$2^{O(k^3)}$	n	[Bod96] ×
$3.67k$	$2^{3.6982k} k^3 \approx 12.98^k$	$n^3 \log^4 n$	[Ami01]
$4k$	$2^{4.38k} k \approx 20.82^k$	n^2	[Ami01]
$4.5k$	$2^{3k} k^{3/2} \approx 8^k k^{3/2}$	n^2	[Ami01]
$5k$	$2^{O(k)} = 81^k k^2$	$n \log n$	[Ree92] ×
$O(k\sqrt{\log k})$	1	$n^{O(1)}$	[FHL05]
$O(k \log k)$	$k^5 \log k$	$n^3 \log^4 n$	[Ami01]

Calculer une décomposition arborescente optimale (cf. ligne 1 du théorème 11) pour de grands graphes de largeur arborescente ≥ 10 en moins de 24h reste un challenge (voir [DK07, ZH09] pour des études expérimentables). Le meilleur algorithme (non FPT) calculant la largeur arborescente d'un graphe à n sommets est en $O(1.8899^n)$ [FTKV08].

Notations.

- Pour tout ensemble de sommets X d'un graphe G , on note $G[X]$ le sous-graphe de G induit par les sommets de X . Plus formellement, $V(G[X]) = X$ et $E(G[X]) = \{\{u, v\} \in E(G) : u, v \in X\}$.
- Pour tout arbre enraciné T et sommet X de T , on note T_X le sous-arbre induit par les descendants X dans T , X étant considéré comme un descendant de lui-même.
- Par abus, on notera $G[T_X]$ le sous-graphe de G induit par tous les sommets de G contenus dans les sacs de T_X . Plus formellement, $G[T_X] = G[\cup_{X \in V(T_X)} X]$.

2.8.2 Un exemple simple : 3-COLORATION

Pour illustrer plus simplement la technique de programmation dynamique sur une décomposition arborescente on choisit d'abord un problème de décision. On rappelle la définition du problème.

3-COLORATION :

Instance: un graphe G

Question: est-ce que G admet une 3-coloration ?

Ce problème est NP-complet, même si G est un graphe planaire¹¹.

Théorème 12 3-COLORATION *pour tout graphe G à n sommets peut être résolu (et on peut donner la coloration correspondante) en temps $2^{O(\text{tw}(G))} \cdot n \log n$.*

Ce problème est donc FPT en la largeur arborescente du graphe.

Preuve. Soit T une décomposition arborescente de G calculée en temps $2^{O(\text{tw}(G))} \cdot n \log n$ et de largeur $\omega \leq 5\text{tw}(G)$. On suppose que T a pour racine le sac noté R . Pour chaque sac X de T , on va calculer récursivement à partir des feuilles deux ensembles, notés $\text{COL}(X)$ et $\text{SOL}(X)$, et définis comme suit (cf. figure 2.26 pour une illustration) :

- $\text{COL}(X)$ est l'ensemble des 3-colorations possibles pour le sous-graphe $G[X]$.
- $\text{SOL}(X)$ est l'ensemble des 3-colorations de $G[X]$ qu'il est possible d'étendre à une 3-coloration du sous-graphe $G[T_X]$.

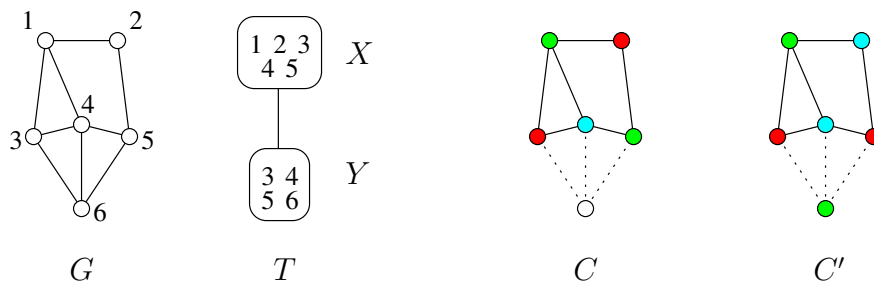


FIGURE 2.26 – Une décomposition arborescente T pour G . Une 3-coloration C de $G[X]$ qui n'est pas dans $\text{SOL}(X)$. Et une autre 3-coloration $C' \in \text{SOL}(X)$.

Une coloration de X est une fonction $C : X \rightarrow \{1, 2, 3\}$ des sommets de X qui peut être représentée par une suite entière de longueur $|X|$ où le i -ème élément est la couleur du i -ème sommet de X (on peut supposer que les sommets de G sont au départ étiquetés de manière

11. Le problème de la k -coloration des graphes planaire est triviale pour $k = 1, 2, 4$. Malheureusement, pas pour $k = 3$.

unique de 1 à n). Donc $\text{COL}(X)$ peut être vu comme un ensemble de suites de $|X| \leq \omega + 1$ entiers de $\{1, 2, 3\}$. On a bien évidemment que $|\text{SOL}(X)| \leq |\text{COL}(X)| \leq 3^{|X|} \leq 3^{\omega+1}$, car $\text{SOL}(X) \subseteq \text{COL}(X)$.

Bien sûr, G est 3-coloriable si et seulement si $\text{SOL}(R) \neq \emptyset$, car $G[T_R]$ est ni plus ni moins que le graphe G lui-même. On cherche donc à calculer $\text{SOL}(R)$.

Remarquons que si X est une feuille, alors $G[T_X] = G[X]$ et donc $\text{SOL}(X) = \text{COL}(X)$ qui se calcule facilement en testant les $3^{|X|}$ suites possibles. Il faut un temps $O(|X|^2)$ pour vérifier qu'une suite est bien une 3-coloration de $G[X]$ qui possède au plus $|X|^2$ arêtes¹². Donc pour chaque feuille X , $\text{SOL}(X)$ se calcule en un temps $O(3^{|X|} \cdot |X|^2) = 2^{O(\omega)}$.

Rappelons que $3^x = 2^{O(x)}$, car pour x assez grand, il existe une constante $c > 0$ telle que $3^x \leq 2^{cx} = (2^c)^x$ (prendre $c = 2$ par exemple).

Supposons maintenant que X a pour fils les sacs Y_1, \dots, Y_k et que les ensembles $\text{SOL}(Y_i)$ ont été calculés pour chaque Y_i . On dit qu'une coloration $C \in \text{COL}(X)$ est « compatible » avec Y_i s'il existe une coloration $C_i \in \text{SOL}(Y_i)$ telle que $C(u) = C_i(u)$ pour tout sommet $u \in X \cap Y_i$.

On peut vérifier assez facilement que $C \in \text{SOL}(X)$ si et seulement si elle est compatible avec tous ses fils. En effet, si $C \in \text{SOL}(X)$, alors, par définition de $\text{SOL}(X)$, il doit exister des colorations $C_i \in \text{SOL}(Y_i)$ telles que $C(u) = C_i(u)$ pour tout $u \in X \cap Y_i$, et donc C est compatibles avec tous ses fils. Inversement, si une coloration $C \in \text{COL}(X)$ est compatible avec tous ses fils, alors $C \in \text{SOL}(X)$ car il ne peut avoir d'arête entre $Y_i \setminus X$ et $X \setminus Y_i$.

On peut calculer $\text{SOL}(X)$ en testant la compatibilité de X avec tous ses fils comme suit :

Algorithme $\text{SOL}(X)$

1. Calculer $S := \text{COL}(X)$.

2. Pour chaque fils Y_i de X faire :

Pour toutes les colorations $C \in S$ et $C_i \in \text{SOL}(Y_i)$, si $C(u) = C_i(u)$ pour tout $u \in X \cap Y_i$, alors $S := S \setminus \{C\}$.

3. Renvoyer S .

On peut tester la compatibilité de C et C_i en temps $O(|X \cap Y_i|)$ une fois que l'intersection a été calculée. L'intersection se calcule en temps $O(|Y_i| + |X|)$, en supposant que tous les sacs de T ont été préalablement triés en temps $O(|V(T)| \cdot \omega \log \omega)$. Donc, en supposant que cette étape de est déjà faite, l'algorithme de calcul de $\text{SOL}(X)$ a une complexité de :

$$O(\underbrace{3^{|X|} \cdot |X|^2}_{2^{O(\omega)}}) + \sum_{i=1}^k \left(O(\underbrace{|X| + |Y_i|}_{2\omega}) + \underbrace{|\text{COL}(X)|}_{3^{\omega+1}} \cdot \underbrace{|\text{SOL}(Y_i)|}_{3^{\omega+1}} \cdot O(\underbrace{|X \cap Y_i|}_{\omega}) \right)$$

12. En fait, $G[X]$ ne peut avoir plus de $|X|^2/3$ arêtes, nombre qui est atteint pour un graphe tri-parties complet $K_{n/3, n/3, n/3}$.

$$\leq 2^{O(\omega)} + \deg_T(X) \cdot 2^{O(\omega)} .$$

Au total, la complexité pour calculer $\text{SOL}(R)$ est (le premier terme est pour le tri de tous les sacs de T) :

$$O(|V(T)| \cdot \omega \log \omega) + \sum_{X \in V(T)} (2^{O(\omega)} + \deg_T(X) \cdot 2^{O(\omega)}) = 2^{O(\omega)} \cdot |V(T)| .$$

Bien sûr, la constante cachée dans le dernier terme $2^{O(\omega)}$, n'est pas la même que dans les termes précédant.

Notons qu'une décomposition arborescente d'un graphe à n sommets n'a pas forcément $O(n)$ sacs. On peut par exemple dupliquer à l'infini le même sac sans violer les conditions de la définition. Cependant, il n'est pas difficile de voir que si pour toute arête $\{X, Y\}$ de T , $X \not\subseteq Y$ (ce qui, le cas échéant, peut être détecté et corrigé facilement en contractant l'arête en question), alors T possède au plus n sacs. On parle alors de décomposition « réduite ». Toutes les décompositions construites dans le théorème 11 sont réduites. Donc $|V(T)| \leq n$.

La complexité finale est donc (on rappelle que $\omega \leq 5\text{tw}(G)$) :

$$2^{O(\text{tw}(G))} \cdot n \log n + 2^{O(\omega)} \cdot n = 2^{O(\text{tw}(G))} \cdot n \log n .$$

□

2.8.3 COUVERTURE PAR SOMMETS pour *treewidth* bornée

On va maintenant considérer un problème avec un paramètre : COUVERTURE PAR SOMMETS (qui est bien plus simple que ENSEMBLE DOMINANT).

Théorème 13 COUVERTURE PAR SOMMETS *de taille minimum pour tout graphe G à n sommets peut être résolu (et on peut donner la couverture correspondante) en temps $2^{O(\text{tw}(G))} \cdot n \log n$.*

Ce problème est donc FPT en la largeur arborescente du graphe.

Preuve. Comme précédemment, l'algorithme consiste à d'abord calculer une décomposition arborescente de G de largeur $\omega \leq 5\text{tw}(G)$ en temps $2^{O(\text{tw}(G))} \cdot n \log n$, puis de résoudre le problème directement sur T en temps $2^{O(\omega)} \cdot n$, ce qui fait un total de $2^{O(\text{tw}(G))} \cdot n \log n$.

...

[A FINIR]

□

En fait toute propriété de graphe exprimable par une formule Φ de logique monadique du second ordre¹³ est FPT en la largeur arborescente du graphe. Les problèmes CHEMIN HAMILTONIEN, k -COLORATION (pour k fixé), ou POSSÉDER H COMME SOUS-GRAPHE (pour H fixé), qui sont NP-complets, sont des exemples de problèmes exprimables par de telles formules. Ce résultat est connu sous le nom de Théorème de Courcelle [Cou90], dont on peut lire aussi la preuve dans [FG06].

Plus précisément, étant données une décomposition arborescente (réduite) d'un graphe G à n sacs et de largeur ω , et une formule Φ monadique du second ordre, on peut tester $\Phi(G) = \text{VRAI}$ en temps $C(\Phi, \omega) \cdot n$ où $C(\Phi, \omega)$ est une constante indépendante de n . C'est donc linéaire en n lorsque Φ et ω sont fixés. Malheureusement, cette constante dépend de manière super-exponentielle en le nombre $h(\Phi)$ d'alternances de quantificateurs $\forall \dots \exists \dots$ de la formule :

$$C(\Phi, \omega) \approx \text{tour}(2, h(\Phi))^\omega .$$

En pratique un algorithme construit « à la main » sera bien plus efficace que la méthode générale pour une formule Φ quelconque.

2.8.4 Application aux graphes planaires

Les deux prochaines sections sont consacrées au problème ENSEMBLE DOMINANT de taille k . Rappelons qu'on ne sait pas si ce problème est FPT en k dans le cas général. Pour les graphes planaires, on a déjà vu un algorithme FPT en k (voir la section 2.5.3). On va ici construire un algorithme très différent, de complexité linéaire en n au lieu de quadratique. Pour cela, on va se servir des décompositions arborescentes des graphes planaires et relier la largeur arborescence et la taille des ensembles dominants. Bien évidemment, les graphes planaires ne sont pas, en général, de largeur arborescente bornée.

Théorème 14 ENSEMBLE DOMINANT de taille k pour les graphes planaires à n sommets peut être résolu (et un tel ensemble peut être construit) en temps $O(3^{9k} k^2 \cdot n)$.

Notons que $3^{9k} = (19\,683)^k \approx 2^{14.26k}$. On aura besoin du résultat suivant, qui s'appuie (mais pas seulement) sur la méthode de programmation dynamique similaire à celle développée dans les sections 2.8.3 et 2.8.2.

Lemme 4 ([VRBR09]) *Étant donnée une décomposition arborescente de largeur ω d'un graphe G à n sommets, on peut résoudre ENSEMBLE DOMINANT de taille k pour G en temps $O(3^\omega \omega^2 \cdot n)$.*

13. Il s'agit de formules avec des quantificateurs \forall et \exists pouvant porter sur des ensembles de sommets ou d'arêtes. Typiquement, la 3-COLORATION d'un graphe G s'exprime par la formule $\Phi(G) = \exists X, Y, Z, X \cup Y \cup Z = V(G), X \cap Y = X \cap Z = Y \cap Z = \emptyset, \forall xy \in E(G), \neg(x \in X \wedge y \in X) \wedge \neg(x \in Y \wedge y \in Y) \wedge \neg(x \in Z \wedge y \in Z)$. Ici, $X \cap Y = \emptyset$ est un raccourci de la formule $\neg(\exists x, x \in X \wedge x \in Y)$.

Dans [LMS11] il a été montré que si ENSEMBLE DOMINANT de taille k peut être résolu pour tout graphe G à n sommets de largeur arborescente ω en temps $(3 - \epsilon)^\omega \cdot n^{O(1)}$ pour un certain $\epsilon > 0$, alors SAT peut être résolue en temps $(2 - \delta)^n \cdot n^{O(1)}$ pour un certain $\delta > 0$, n étant le nombre de variables de la formule. Comme personne ne croit à cette dernière affirmation, tout le monde pense que « 3 » est le meilleur exposant possible pour ENSEMBLE DOMINANT pour les graphes de largeur arborescente ω .

La suite de la section est consacrée à la preuve du théorème 14.

Lemme 5 *Soit G un graphe planaire à n sommets possédant un arbre couvrant de hauteur h . Alors en temps $O(nh)$ on peut calculer une décomposition arborescente de largeur $\leq 3h$. En particulier, $\text{tw}(G) \leq 3D$ où D est le diamètre de G .*

Preuve. Soit G un graphe planaire dessiné dans le plan et soit S un arbre couvrant de hauteur h . On commence par trianguler le graphe G en ajoutant autant que possible des arêtes aux faces de G tout en préservant la planarité. Il est possible de le faire en temps $O(n)$, G ayant au plus $O(n)$ faces et $O(n)$ arêtes.

On va en fait calculer une décomposition arborescente du graphe triangulé. Pour obtenir celle de G , il suffira de supprimer les arêtes ajoutées. Cela restera une décomposition arborescente de G . Pour simplifier, on appelle G le graphe triangulé.

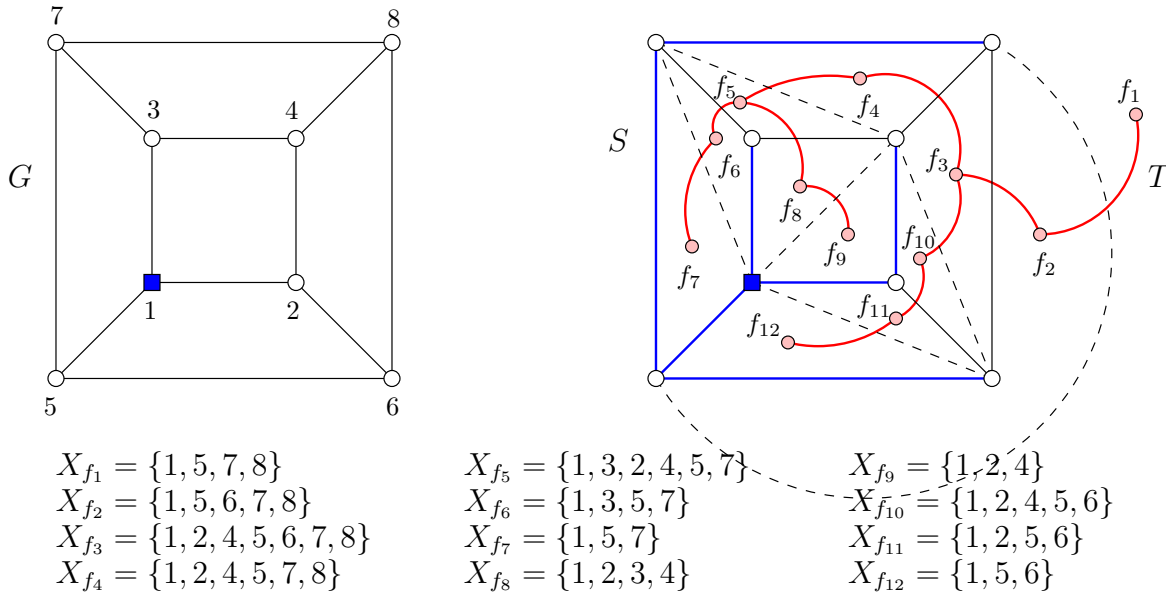


FIGURE 2.27 – Construction d’une décomposition arborescente T (en rouge) de largeur $\leq 3h$ pour un graphe planaire G à partir d’un arbre couvrant S (en bleu) de profondeur $h = 3$. En fait, $\text{tw}(G) = 3$.

À chaque face f de G on associe un sac X_f formé des sommets des trois chemins dans S menant d’un bord de f à la racine de S . Ces trois chemins comprennent au plus $3h$ arêtes.

Bien évidemment, $|X_f| \leq 3h + 1$. L'arbre T de la décomposition est défini par la collection de sacs $\{X_f\}_f$, et sacs X_f et $X_{f'}$ sont adjacents si et seulement si les bords des faces f et f' partagent une arête de $G \setminus S$. Dit autrement T est le dual de G dans lequel les arêtes intersectant S sont supprimées.

Le calcul de T se fait en temps $O(nh)$, il y a au plus $O(n)$ sacs (car $O(n)$ faces), et chaque sac se construit en temps $O(h)$ une fois que S a été construit.

On va montrer que T est bien une décomposition arborescente pour G . Montrons que T n'a pas de cycle. Supposons qu'il existe un cycle $C = X_{f_1}, \dots, X_{f_k}$ dans T . Alors il forme, dans le dual de G , une courbe partageant les sommets de G en ceux qui sont à l'intérieur et ceux qui sont à l'extérieur de C . Il est clair que chacune des deux régions ainsi formées contient au moins un sommet (au moins une arête $\{x, y\}$ de G coupe une arête $\{f_i, f_{i+1}\}$ du dual, cf. figure 2.28). Or toutes les arêtes de G traversant cette courbe ne sont pas dans S par définition de C : contradiction avec le fait que S couvre tous les sommets de G .

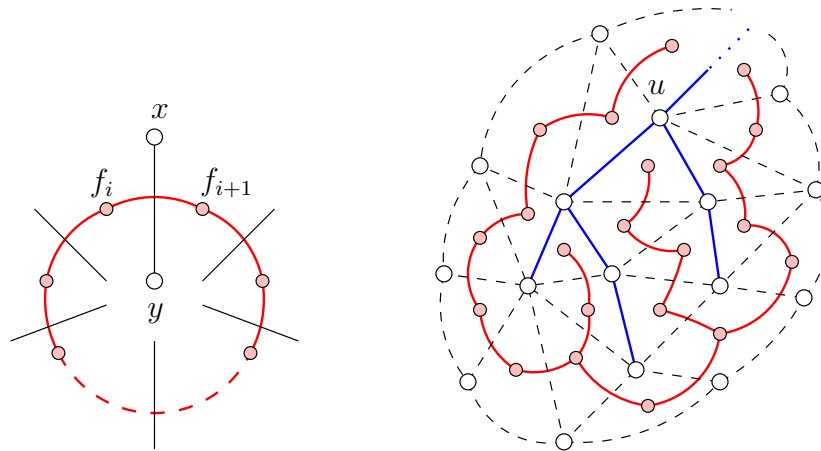


FIGURE 2.28 – T ne contient pas de cycle, et les sacs contenant u induisent un sous-arbre de T .

On observe aussi que T est un graphe connexe. En effet, dans le dual on peut connecter toute paire de faces, sans intersecter S , en passant, par exemple, par la face extérieure (sinon c'est S qui comprendrait un cycle). Donc T est bien un arbre.

Clairement T couvre tous les sommets et les arêtes de G . Reste à montrer que la 3e propriété de la définition des décompositions arborescentes est respectée. Par construction, les sacs contenant un sommet u donné sont les faces dont au moins un sommet du bord est un descendant de u dans S (u compris). Dans le dual, ces faces correspondent à un parcours Eulérien du sous-arbre S_u , et induisent un sous-arbre de T , comme suggéré par la figure 2.28. \square

Lemme 6 Soit G un graphe de diamètre D possédant un ensemble dominant de taille k . Alors $D \leq 3k - 1$.

Preuve. Soit P un plus court chemin de longueur D dans G , et S un ensemble dominant avec $|S| = k$. On dit qu'un sommet $s \in S$ domine un sommet x si $x \in N[s] = B_G(s, 1)$. Bien sûr, S doit dominer tous les sommets de G et en particulier tous ceux de P . La remarque principale est que tout sommet de S domine au plus trois sommets de P .



FIGURE 2.29 – Chaque sommet $s_i \in S$ domine au mieux trois sommets d'un plus court chemin P .

En effet, si un sommet s domine plus de trois sommets de P , alors P ne serait pas un plus court chemin (il y a deux cas à considérer : $s \in P$ et $s \notin P$, cf. figure 2.29). Donc si $|V(P)| > 3|S|$, alors un sommet de P ne serait pas dominé. Donc on a à montrer que $D \leq 3k - 1$, car $|V(P)| = D + 1 \leq 3|S| = 3k$. \square

On déduit des deux lemmes précédant un algorithme simple qui détermine si un graphe planaire G à n sommets possède un ensemble dominant de taille k .

Algorithme DOM-PLANAIRE1(G, k)

1. Calculer en temps $O(n)$ un arbre couvrant en largeur d'abord, et soit h sa hauteur.
 2. Si $h > 3k - 1$, renvoyer FAUX (lemme 6).
 3. Calculer une décomposition arborescente T de largeur $\omega \leq 3h$ (lemme 5).
 4. Résoudre grâce à T , ENSEMBLE DOMINANT de taille k en temps $O(3^\omega \omega^2 n)$ (lemme 4).
-

Pour l'étape 2, en effet par le lemme 6, si G possède un ensemble dominant de taille k , alors $h \leq D \leq 3k - 1$. On note que $\omega \leq 3h \leq 3 \times (3k - 1) < 9k$. Donc la complexité totale de l'algorithme est $O(n + nh + 3^{9k} \cdot n) = O(3^{9k} k^2 \cdot n)$, car $nh = O(nk) = O(3^{9k} k^2 \cdot n)$, ce qui termine la preuve du théorème 14.

2.8.5 Amélioration pour les graphes planaires

On peut encore réduire la complexité de l'algorithme précédant, en diminuant la fonction $f(k) = 2^{O(k)}$, grâce aux deux lemmes suivants. Le premier (lemme 7) est très profond. Sa preuve complète sort très largement du cadre de ce cours. On admettra donc ce résultat qui permet de justifier l'analyse de l'algorithme (on l'utilise seulement dans la preuve du lemme 8), mais qui ne change pas l'algorithme lui-même.

Lemme 7 ([RST94, pp. 346] et [GT10]) *Soit G un graphe planaire. Si $\text{tw}(G) > (9s + 3)/2$, alors G contient une grille $s \times s$ comme mineur.*

Une version plus faible de ce résultat, avec la condition $\text{tw}(G) = \Omega(s^2)$, avait été présentée dix ans plus tôt dans [RS84]. Il est conjecturé dans [GUM10] que le terme « $(9s + 3)/2$ » du lemme 7 peut être remplacé par $2s + o(s)$, et il est démontré que c'est au moins $\lfloor 3s/2 - 1 \rfloor$. Dans [RST94] il est également démontré un résultat beaucoup plus général : si G (graphe quelconque) ne contient pas un graphe planaire H comme mineur, alors

$$\text{tw}(G) \leq 20^{4(|V(H)|+2|E(H)|)^5}. \quad (2.1)$$

On en déduit par exemple que les graphes sans K_4 comme mineur sont de largeur arborescente au plus $2^{4(4+2\cdot 6)^5} \approx 2^{400000}$. En fait ces graphes sont exactement les sous-graphes des graphes série-parallèles et ont une largeur arborescente au plus 2.

Lemme 8 *Soit G un graphe planaire possédant un ensemble dominant de taille k . Alors $\text{tw}(G) \leq 13.5\sqrt{k} + 15$.*

Preuve. Soit ℓ le plus grand entier tel que G possède une grille $\ell \times \ell$ comme mineur. Éventuellement, $\ell = 1$, si par exemple, G n'a pas de cycle de longueur ≥ 4 .

On considère un model de cette grille dans G . Il est composé de sous-graphes connexes disjoints deux à deux, qu'on appellera super-nœuds, et interconnectés comme une grille. Notons que dans G il peut avoir d'autres arêtes entre ces super-nœuds, des diagonales par exemple.

Sans perte de généralité on va supposer que tout sommet de la composante connexe G' de G contenant la grille $\ell \times \ell$ appartient à un super-nœud. Dit autrement, les super-nœuds forment une partition de $V(G')$. C'est le cas de l'exemple de la figure 2.30. Si cela n'est pas le cas, on peut construire un arbre T enraciné en un sommet d'un super-nœud et couvrant G' . Et, on rattache à tout super-nœud X les sommets de G' qui ne sont dans aucun super-nœud qui ont comme plus proche ancêtre un sommet de X .

Soit S un ensemble dominant de taille k pour G . On dira que $s \in S$ domine un super-nœuds s'il domine au moins un sommet de ce super-nœud. Bien sûr, S doit dominer tous les sommets de G , en particulier tous les super-nœuds.

La remarque importante, liée à la planarité de G , est que tout sommet de S domine au plus 9 super-nœuds internes de la grille. Pour le montrer, on va supposer que $\ell > 2$, puisque sinon la question ne se pose pas, le model n'a pas de super-nœud interne.

Remarquons qu'à tout cycle C de la grille on peut construire un cycle de G qui utilise les arêtes entre les super-nœuds de C et qui utilise un chemin à l'intérieur de chaque super-nœud traversé. Rappelons que les super-nœuds sont connexes et forment une partition de $V(G')$. Soit $s \in S$. Si $s \notin V(G')$, alors s domine aucun super-nœud. Sinon, $s \in V(G')$ et deux cas de figure se présentent (cf. figure 2.32) :

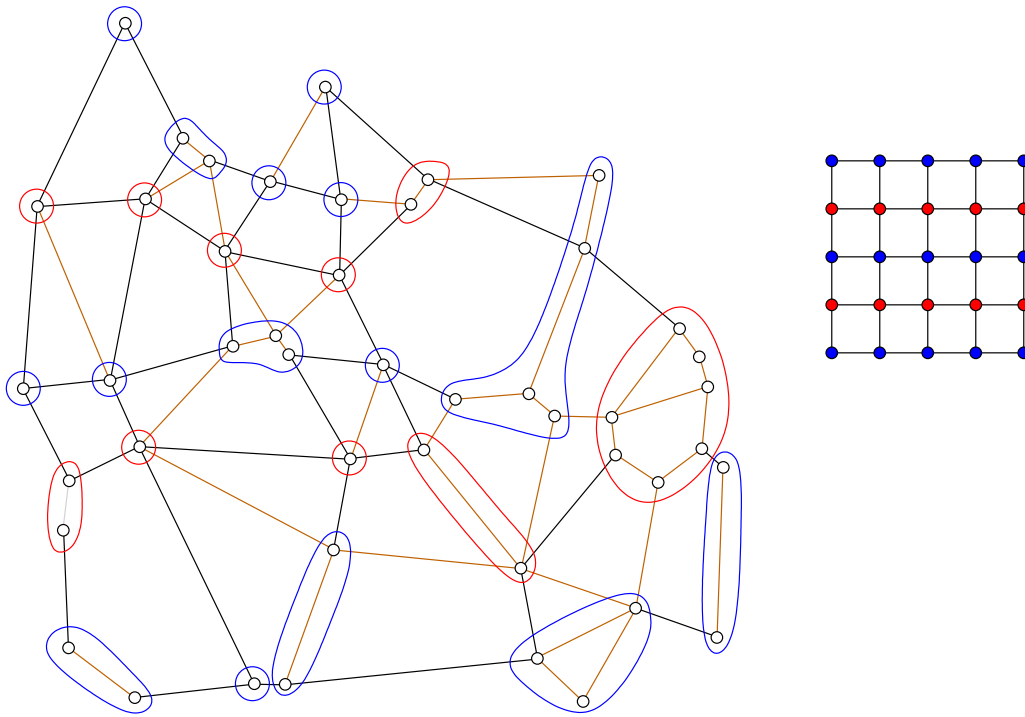


FIGURE 2.30 – Une grille 5×5 mineur d'un graphe de Gabriel à 45 sommets. Les sommets sont des points du plan, et deux points u, v sont adjacents ssi l'intérieur du disque de diamètre $|uv|$ ne contient aucun autre point.

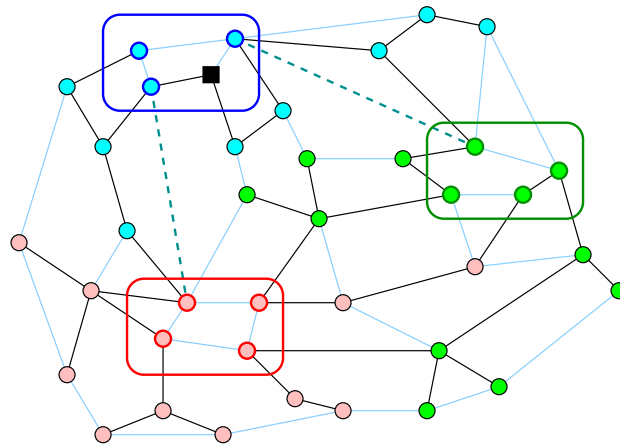
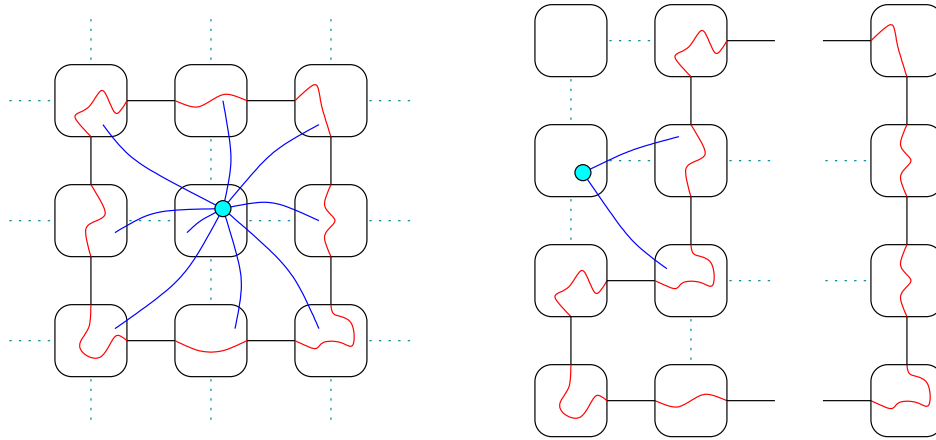


FIGURE 2.31 – Construction d'une partition à partir des super-nœuds et d'un arbre couvrant.

Cas 1 : s appartient à un super-nœud (x, y) interne. Dans ce cas on considère le cycle C de G correspondant au cycle de la grille passant par les 8 voisins de (x, y) . Il délimite une

FIGURE 2.32 – Super-nœuds internes dominés par un sommet $s \in S$.

région où les voisins de s doivent se situer. Cette région comprend au plus 9 super-nœuds internes.

Cas 2 : s appartient à un super-nœud (x, y) du bord. Dans ce cas on considère le cycle C de G correspondant au cycle du bord de la sous-grille obtenue en supprimant (x, y) de la grille originale et puis éventuellement les sommets de degré 1 de la sous-grille (si (x, y) est sur la 2e ligne ou 2e colonne). Il s'agit bien d'un cycle car $\ell \geq 3$. Là encore, C délimite une région où les voisins de s doivent se situer. Cette région comprend tous les super-nœuds du bord mais au plus 3 super-nœuds internes (cela peut être 1 ou 2 comme dans la figure 2.32).

Le mineur possède $(\ell - 2)^2$ super-nœuds internes. Donc $(\ell - 2)^2 \leq 9|S| = 9k$ puisque sinon un super-nœud (et donc certains sommets de G) ne seraient pas dominés par S . Il suit que $\ell \leq 3\sqrt{k} + 2$. On sait, par le choix de ℓ , que G ne possède pas de grille $(\ell + 1) \times (\ell + 1)$ comme mineur. D'après le lemme 7 (sa contraposée avec $s = \ell + 1$) :

$$\text{tw}(G) \leq \frac{1}{2} \cdot 9(\ell + 1) + 3 = \frac{9}{2}\ell + 6 \leq \frac{27}{2}\sqrt{k} + 15 = 13.5\sqrt{k} + 15 .$$

□

On en déduit donc l'algorithme suivant :

Algorithme DOM-PLANAIRE2(G, k)

1. Soit $t := 13.5\sqrt{k} + 15$.
 2. Déterminer en temps $2^{O(t)} \cdot n \log n$ si $\text{tw}(G) > t$ ou sinon construire une décomposition arborescente T de largeur $\omega \leq 5t$ (théorème 11).
 3. Si $\text{tw}(G) > t$, alors renvoyer FAUX (lemme 8).
 4. Résoudre ENSEMBLE DOMINANT grâce à T en temps $O(3^\omega \omega^2 \cdot n)$ (lemme 4).
-

Comme $\omega = O(t) = O(\sqrt{k})$, la complexité de l'algorithme est $2^{O(\sqrt{k})} \cdot n \log n$. On a donc montrer que :

Théorème 15 ENSEMBLE DOMINANT de taille k pour les graphes planaires à n sommets peut être résolu (et un tel ensemble peut être construit) en temps $2^{O(\sqrt{k})} \cdot n \log n$.

Il faut remarquer qu'en utilisant l'algorithme donnant une approximation à un facteur 5 sur la largeur arborescente, et la résolution en temps $O(3^\omega \omega^2 \cdot n)$ pour ENSEMBLE DOMINANT, on obtient, pour le terme $f(k)$ exponentiel en \sqrt{k} la valeur suivante :

$$3^\omega \leq 3^{5t} \approx 2^{107\sqrt{k}} \approx (10^{32})^{\sqrt{k}} = \underbrace{1000 \dots 000}_{32}^{\sqrt{k}}.$$

L'algorithme FPT pour ENSEMBLE DOMINANT dans les graphes planaires ayant la plus faible fonction $f(k)$, tout en étant au plus cubic, a une complexité en temps de $O(35860^{\sqrt{k}} \cdot k + n^3)$ [FT06]. Ce résultat est obtenu en combinant les techniques précédentes avec celle du noyau vu au paragraphe 2.6. Il existe aussi un algorithme de complexité $4040^{\sqrt{k}} \cdot n^{O(1)}$ mais le polynôme $n^{O(1)}$ est de degré relativement grand [Dor10].

2.9 Remarques finales

L'algorithme en $O(35860^{\sqrt{k}} \cdot k + n^3)$ doit être comparé à celui en $O(8^k \cdot n^2)$ (cf. le théorème 6), qui peut en fait être amélioré en $O(8^k \cdot n)$ (voir [AFF⁺05]). Ce dernier est plus rapide lorsque k est « assez petit », et surtout plus simple à programmer¹⁴. Le goulot d'étranglement pour la technique des décompositions arborescentes reste le calcul de la décomposition.

Enfin, signalons que parfois des algorithmes FPT sont utilisés à la place d'algorithmes polynomiaux. Par exemple, un algorithme en $O(2^k \cdot n)$ peut être bien plus rapide, lorsque k est assez petit devant n , qu'un algorithme en $O(n^6)$, soit $k < 5 \log n$. Pour $n = 1000$ par exemple, l'algorithme exponentiel sera plus rapide pour $k \leq 50$, valeur déjà intéressante. Notez aussi qu'exécuté sur un ordinateur cadencé à une fréquence d'1 GHz (soit 10^9 instructions/s), l'algorithme en n^6 prendra plus de 30 ans.

Un exemple classique est le calcul du flût maximum d'un graphe orienté valué, disons avec des capacités entières. L'algorithme de Ford-Fulkerson (basé sur les « chaînes améliorantes »), très simple à implémenter, est de complexité $O(mk)$ où k est la valeur du flût maximum. Potentiellement, k peut être exponentiel en n , et même non borné. Cet algorithme est FPT en k . L'algorithme d'Edmonds-Karp, une variante basé sur la recherche de chaînes améliorantes qui sont des plus courts chemins, a une complexité polynomiale en $O(nm^2)$, indépendante de k donc. Et pour $k = o(nm)$ (ce qui est beaucoup en pratique), Ford-Fulkerson sera toujours plus rapide qu'Edmonds-Karp.

14. En fait, pour $n = 32, 64, 128$ le premier algorithme devient meilleur seulement lorsque $k \geq 19, 17, 14$ respectivement. À partir de $n = 1024$, il est toujours meilleur.

Bibliographie

- [ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2) :277–284, April 1987.
- [AFF⁺05] Jochen Alber, Hongbing Fan, Michael R. Fellows, Henning Fernau, Rolf Niedermeier, Fran Rosamond, and Ulrike Stege. A refined search tree technique for Dominating Set on planar graphs. *Journal of Computer and System Sciences*, 71(4) :385–405, 2005.
- [AKFLS07] Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3) :411–430, 2007.
- [AL91] Stefan Arnborg and Jens Lagergren. Finding minimal forbidden minors using a finite congruence. In *18th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 510 of Lecture Notes in Computer Science, pages 532–543. Springer, July 1991.
- [Ami01] Eyal Amir. Efficient approximation for triangulation of minimum treewidth. In *17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 7–15. Morgan Kaufmann, August 2001.
- [AP87] Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic and Discrete Methods*, 7(2) :305–314, April 1987.
- [APC90] Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80(1) :1–19, February 1990.
- [BFL⁺09] Hans Leo Bodlaender, Fedor V. Fomin, Daniel Lokshantov, Eelko Penninx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *50st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 629–638. IEEE Computer Society Press, October 2009.
- [Bod96] Hans Leo Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6) :1305–1317, 1996.
- [CFKX05] Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization : Lower bounds and upper bounds on kernel size. In *22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3404 of Lecture Notes in Computer Science, pages 269–280. Springer, February 2005.
- [CKX06] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for Vertex Cover. In R. Kráľovic and P. Urzyczyn, editors, *31st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 4162 of Lecture Notes in Computer Science, pages 238–249. Springer, August 2006.

- [Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1) :12–75, March 1990.
- [DK07] Alex P. Dow and Richard E. Korf. Best-first search for treewidth. In *22nd National Conference on Artificial Intelligence (SPAA)*, volume 2, pages 1146–1151. AAAI Press, July 2007.
- [Dor10] Frederic Dorn. Dynamic programming and planarity : Improved tree-decomposition based algorithms. *Discrete Applied Mathematics*, 158(7) :800–808, April 2010.
- [FG06] Jörg Flum and Martin Grohe. *Parametrized Complexity Theory*. Springer, 2006.
- [FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5) :Article No. 25, August 2009.
- [FHL05] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 563–572. ACM Press, May 2005.
- [FL89] Michael R. Fellows and Michael A. Langston. On search, decision and the efficiency of polynomial-time algorithms. In *21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 501–512. ACM Press, May 1989.
- [FT06] Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs : Branch-width and exponential speed-up. *SIAM Journal on Computing*, 36(2) :281–306, 2006.
- [FTKV08] Fedor V. Fomin, Ioan Todinca, Dieter Kratsch, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM Journal on Computing*, 38(3) :1058–1079, 2008.
- [GL06] Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In *6th Italian Conference on Algorithms and Complexity (CIAC)*, volume 3998 of Lecture Notes in Computer Science, pages 6–17. Springer, May 2006.
- [GMC09] Andrei Gagarin, Wendy Myrvold, and John Chambers. The obstructions for toroidal graphs with no $K_{3,3}$'s. *Discrete Mathematics*, 309 :3625–3631, 2009.
- [GT10] Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branch-width with respect to the largest grid minor size. In *21st Annual International Symposium on Algorithms and Computation (ISAAC)*, volume 6507 of Lecture Notes in Computer Science, pages 85–96. Springer, December 2010.
- [GUM10] Alexander Grigoriev, Natalya Usotskaya, and Bert Marchal. On planar graphs with large tree-width and small grid minors. *Electronic Notes in Discrete Mathematics*, 32 :35–42, March 2010.

- [Khu02] Samir Khuller. The vertex cover problem. *ACM SIGACT News - Algorithms Column*, 33(2) :31–33, June 2002.
- [KKM09] Ken-ichi Kawarabayashi, Stephan Kreutzer, and Bojan Mohar. Linkless and flat embeddings in 3-space in quadratic time. Technical Report 1070, University of Ljubljana, Slovenia, January 2009.
- [LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *22nd Symposium on Discrete Algorithms (SODA)*, pages 777–789. ACM-SIAM, January 2011.
- [Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications - 31, 2006.
- [NP85] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of subgraph problem. *Commentationes Mathematicae Universatis Carolinae*, 26(2) :415–419, 1985.
- [Ree92] Bruce A. Reed. Finding approximate separators and computing treewidth quickly. In *24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 221–228. ACM Press, 1992.
- [RS84] Neil Robertson and Paul D. Seymour. Graph width and well-quasi-ordering : a survey. In Adrian J. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory*, pages 399–406. Academic Press, 1984.
- [RS95] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1) :65–110, 1995.
- [RS04] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner’s conjecture. *Journal of Combinatorial Theory, Series B*, 92(2) :325–357, 2004.
- [RST94] Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2) :323–348, 1994.
- [RST95] Neil Robertson, Paul D. Seymour, and Robin Thomas. Sach’s linkless embedding conjecture. *Journal of Combinatorial Theory, Series B*, 64(2) :185–227, July 1995.
- [Vik96] Narayan Vikas. An $O(n)$ algorithm for abelian p -group isomorphism and an $O(n \log n)$ algorithm for abelian group isomorphism. *Journal of Computer and System Sciences*, 53(1) :1–9, 1996.
- [VRBR09] Johan M. M. Van Rooij, Hans Leo Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *17th Annual European Symposium on Algorithms (ESA)*, volume 5757 of Lecture Notes in Computer Science, pages 566–577. Springer, September 2009.
- [ZH09] Rong Zhou and Eric A. Hansen. Combining breadth-first and depth-first strategies in searching for treewidth. In *21st International Joint Conference on*

Artificial Intelligence (IJCAI), pages 640–645. Morgan Kaufmann Publishers Inc., July 2009.

3.1 Introduction

Définition c -approximation ... c est le *facteur* d'approximation.

Remarquons qu'une 1-approximation est un algorithme exact polynomial.

Un *vérificateur positif* pour un problème de décision Π est un algorithme \mathcal{V} de complexité polynomiale tel que pour toute instance I :

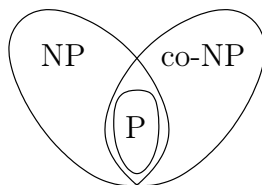
- si I est positive, alors il existe $C \in \{0, 1\}^*$, le certificat, de taille polynomiale en $|I|$ tel que $\mathcal{V}(I, C)$ renvoie VRAI.
- si I est négative, alors pour tout $C \in \{0, 1\}^*$ de taille polynomiale en $|I|$, $\mathcal{V}(I, C)$ renvoie FAUX.

Le mot C qui aide à affirmer que I est une instance positive est un *certificat positif*. On appelle aussi C *preuve*, *solution*, ou encore *témoin* dans le contexte des calculs randomisés. Un vérificateur négatif est défini pareillement en échangeant les termes « positif » et « négatif ».

Prenons un exemple de certificat positif. Un vérificateur pour COUVERTURE PAR SOMMETS de taille $\leq k$ présume que C code un ensemble de sommets. Il vérifie si cet ensemble est bien une couverture par sommets et que sa taille est bien inférieure ou égale à k .

Un certificat négatif pour le problème “ n est premier” est une factorisation non triviale (autre que $1 \cdot n$), ce qui est une preuve que n n'est pas premier.

Il faut observer qu'aucune hypothèse n'est faite sur le temps nécessaire pour *trouver* un certificat. La classe NP est la classe des problèmes possédant un vérificateur positif, alors que la classe co-NP est la classe des problèmes possédant un vérificateur négatif. Les problèmes de P ont trivialement des vérificateurs positifs et négatifs. Il suffit de prendre $C = \varepsilon$ et comme vérificateur l'algorithme polynomial lui-même. Donc, $P \subset NP \cap \text{co-NP}$. La question largement ouverte est de savoir s'il n'y a pas égalité.

FIGURE 3.1 – $P \subset NP \cap \text{co-NP}$.

3.2 Problèmes bien caractérisés

3.2.1 Couplage

Un couplage M de G est *maximal* si pour toute arête $e \in G \setminus M$, $M \cup \{e\}$ n'est pas un couplage. Il est *maximum* (ou de taille maximum) si tout couplage de G à une taille $\leq |M|$ (cf. la figure 3.2).



FIGURE 3.2 – Couplage maximal de taille 2 et couplage maximum de taille 4.

Un couplage maximal, tout comme un ensemble indépendant maximal, se calcule trivialement par un algorithme glouton :

1. $M := \emptyset$
2. Tant qu' $\exists e = (u, v) \in E(G)$, faire $M := M \cup \{e\}$ et $G := G \setminus \{u, v\}$.

Il est possible d'implémenter cet algorithme pour obtenir une complexité linéaire en G , c'est-à-dire $O(n+m)$. [Exercice : comment ?] Trouver un couplage maximum est par contre beaucoup plus difficile. Le problème de décision est NP-complet.

On rappelle qu'un graphe G est *biparti* si l'ensemble de ces sommets se partitionne en deux ensembles indépendants. Dit autrement, G est 2-coloriable, toute arête connectant forcément un sommet d'une partie vers l'autre.

Dans un graphe biparti à n sommets et m arêtes, on peut calculer un couplage maximum en temps $O(m\sqrt{n})$. On peut le faire en utilisant une technique de calcul de flôt maximum. On ajoute une source s connectée à tous les sommets de la partie A et un puit t connecté à tous les sommets de la partie B . On oriente les arêtes de s vers t , et on met une capacité de 1 sur toutes les arêtes. S'il existe un flôt de valeur k , alors G possède un couplage de taille k . Et, inversement, si G possède un couplage de taille k , il existe un flôt de valeur k .

3.2.2 Les problèmes MIN-MAX

Considérons les deux problèmes de décisions suivants :

- (P1) couverture par sommets de taille $\leq k$?
 (P2) couplage de taille $\geq \ell$?

Ces deux problèmes sont dans NP, car ils ont clairement des certificats positifs. Ont-ils des certificats négatifs ?

Théorème 16 (König-Egavary, 1931) *Pour tout graphe biparti G ,*

$$\max_{M \in \mathcal{C}(G)} |M| = \min_{C \in \mathcal{V}(G)} |C|$$

où $\mathcal{C}(G)$ représente l'ensemble des couplages de G , et $\mathcal{V}(G)$ l'ensemble des couvertures par sommets de G .

Supposons que G soit biparti. Si la réponse à (P1) est NON, alors il existe un couplage M de taille $k + 1$. L'ensemble M est un certificat négatif pour (P1). Inversement, si la réponse à (P2) est NON, alors il existe une couverture par sommets C de taille $\ell - 1$. L'ensemble C est un certificat négatif pour (P2).

Donc les restrictions de (P1) et (P2) aux graphes bipartis sont dans $\text{NP} \cap \text{co-NP}$. En fait, ils sont dans P.

Les problèmes dans $\text{NP} \cap \text{co-NP}$ sont dits « bien caractérisés ». De telles relations min-max permettent de prouver que des problèmes sont bien caractérisés. Selon la conjecture $\text{NP} \neq \text{co-NP}$, les problèmes NP-difficiles n'admettent pas de certificats négatifs. Donc montrer qu'un problème admet des certificats positifs et négatifs est généralement un premier pas vers la découverte d'algorithmes polynomiaux.

3.2.3 Comment concevoir un algorithme d'approximation ?

Pour un problème de minimisation, il faut trouver un “bon” minorant, c'est-à-dire le plus grand possible ! Pour un problème de maximisation, c'est le contraire : il faut trouver un majorant le plus petit possible.

Essayons de trouver un algorithme d'approximation pour le problème suivant :

SOUS-GRAPHE SANS CYCLE MAXIMUM :

Instance: un graphe orienté G

Question: trouver un sous-graphe sans cycle de G avec un nombre maximum d'arcs.

Indication : numéroté arbitrairement les sommets de G de 1 à n , et prendre le plus grand des deux ensembles d'arcs suivants : les arcs (i, j) avec $i < j$, et son complémentaire.

Il faut majorer $\text{OPT} : m$. Un des deux ensembles est de taille $\geq m/2$. C'est donc une 1/2-approximation.

3.3 COUVERTURE PAR SOMMETS de taille minimum

Dans cette partie on cherche à déterminer une couverture par sommets de cardinalité minimum. Ce problème est NP-difficile.

3.3.1 Un premier algorithme

Algorithme ALGO1(G)

1. Calculer un couplage M maximal pour G .
 2. Renvoyer l'ensemble C des extrémités des arêtes de M .
-

Proposition 4 *ALGO1 est une 2-approximation pour COUVERTURE PAR SOMMETS de taille minimum.*

Preuve. Il faut montrer trois choses :

1. L'algorithme est polynomial.
2. L'algorithme renvoie une couverture par sommets.
3. La couverture est de taille $\leq 2 \cdot \text{OPT}$.

□

Instance critique : ... un ensemble d'arêtes indépendantes.

3.3.2 Un second algorithme

Le prochain algorithme est un peu plus facile à programmer ppour qu'il s'exécute en temps linéaire. Il nécessite cependant un pré-traitement car il suppose que le graphe d'entrée G est connexe et possède au moins une arête. Dans le cas général, il faut enlever toutes les sommets isolés et appliquer l'algorithme à chacune des composantes connexes, ce qui est facile à traiter.

 Algorithmme ALGO2(G)

1. Calculer pour G un arbre couvrant T en profondeur d'abord.
 2. Renvoyer l'ensemble S des sommets internes de T (c'est-à-dire les sommets qui ne sont pas des feuilles).
-

Proposition 5 ALGO2 est une 2-approximation pour COUVERTURE PAR SOMMETS de taille minimum.

Preuve.

1. L'algorithme est polynomial, il est clairement linéaire.
2. $S = \text{ALGO2}(G)$ est une couverture pour G . En effet, sinon, il existe une arête $\{u, v\}$ de G avec u et $v \notin S$, donc où u et v sont des feuilles de T . C'est impossible si T est un arbre en profondeur : un sommet est visité avant l'autre, disons u , et donc u ne peut devenir une feuille sans qu'on visite v .

[PRENDRE UN EXEMPLE]

3. Montrons que $\text{OPT}(G) \geq |S|/2$. Pour cela, on va construire un couplage M pour G de taille $\geq |S|/2$. Considérons une 2-coloration de $T[S]$. Plus précisément on considère la partition $S = S_0 \cup S_1$ où S_i est l'ensemble à distance $i \bmod 2$ de la racine de T . On choisit $W \in \{S_0, S_1\}$ comme l'ensemble des sommets de S coloriés de la même couleur et dont la couleur est la plus représentée. Clairement, $|W| \geq |S|/2$. On associe pour chaque sommet $u \in W$ une arête, notée $e_u = (u, v)$ où v est un fils arbitraire de u . Notons que e_u existe bien pour tout $u \in W$, car u est un sommet interne de T , donc u a toujours un fils. On vérifie également que l'ensemble d'arêtes $M = \{e_u : u \in W\}$ est bien un couplage de G car si $e_u = (u, v) \neq e_{u'} = (u', v')$, alors on a : $u \neq u'$ (par unicité du choix de l'arête). D'autre part $v = v'$ est impossible sinon ce sommet serait le fils de deux sommets différents : il ne peut avoir deux parents. Conclusion M est un couplage de T donc de G . Précédemment on a vu que $\text{OPT} \geq |M|$ donc $|\text{ALGO2}(G)| = |S| \leq 2|M| \leq 2 \cdot \text{OPT}$. C'est une 2-approximation. \square

Instance critique : ... un chemin de longueur paire où la racine de T est l'une des extrémités.

3.3.3 Technique de précalcul (*color coding*)

Théorème 17 Si G peut être colorié en k couleurs en temps t , alors en temps $t + O(m\sqrt{n})$ on peut calculer une couverture par sommets de taille au plus $(2 - 2/k) \cdot \text{OPT}(G)$.

Remarque : Si le graphe est biparti, alors on peut donner une 2-coloration en temps linéaire. Et par conséquent, le théorème 17 implique un algorithme exact.

Pour cela on aura besoin du lemme suivant, qu'on ne démontrera pas :

Lemme 9 *En temps $O(m\sqrt{n})$ on peut partitionner les sommets d'un graph G en trois ensembles P, Q, R tels que :*

1. *Tous les voisins des sommets de R sont dans P ;*
2. *Il existe une couverture par sommets optimale contenant P ; et*
3. *Toute couverture par sommets a une taille $\geq |P| + \frac{1}{2}|Q|$.*

[FAIRE DESSIN EXEMPLE P,Q,R]

Preuve. Idée [A VOIR] : trouver une couronne R et poser $P = N(R)$ (voir définition 5) et tel que le graphe induit par $Q = V(G) \setminus (R \cup P)$ contient un couplage de taille au moins $|Q|/2$. \square

Preuve du théorème 17. On calcule les ensembles P, Q, R du lemme 9 pour le graphe G . On remarque alors que toute arête de G a soit une extrémité dans P , soit une extrémité dans Q , par définition de R . Autrement dit, si C est une couverture par sommets pour le sous-graphe $G[Q]$, alors, $P \cup C$ est une couverture par sommets pour G .

Soit C une couverture quelconque de $G[Q]$, posons $\alpha = |C|/|Q|$. Évidemment $0 < \alpha \leq 1$ (on peut supposer que $|C| > 0$ et donc que $G[Q]$ contient au moins une arête, sinon on pourrait ajouter Q à l'ensemble R).

Montrons que la couverture $P \cup C$ pour G vérifie $|P \cup C| \leq 2\alpha \cdot \text{OPT}(G)$. En effet, d'après la propriété 3 du lemme 9 :

$$\text{OPT}(G) \geq |P| + \frac{1}{2}|Q| \geq |P| + \frac{1}{2\alpha}|C| \geq \frac{1}{2\alpha}(|P| + |C|) \geq \frac{1}{2\alpha}|P \cup C|. \quad (3.1)$$

D'où $|P \cup C| \leq 2\alpha \cdot \text{OPT}(G)$. On remarque qu'on obtient déjà une 2-approximation en prenant $C = Q$ ($\alpha = 1$).

Cette majoration peut être affinée en construisant une couverture C pour $G[Q]$ plus petite, donc avec $\alpha < 1$. L'idée est d'utiliser un ensemble indépendant de $G[Q]$, disons I . Comme on l'a vu au début du cours, l'ensemble $Q \setminus I$ est une couverture par sommets pour $G[Q]$.

Pour cela on calcule en temps t une k -coloration de G (et donc de $G[Q]$). On pose $I \subset Q$ comme le sous-ensemble de sommets ayant la couleur la plus représentée parmi les sommets de Q . Bien évidemment, I est un ensemble indépendant et $|I| \geq |Q|/k$ puisque la coloration partitionne les sommets de Q en au plus k sous-ensembles disjoints. On en déduit une couverture $C = Q \setminus I$. On a $|C| \leq (1 - 1/k) \cdot |Q|$, et en particulier $\alpha = |C|/|Q| \leq (1 - 1/k)$.

D'après l'équation (3.1), on obtient :

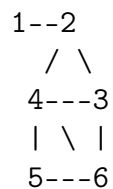
$$|P \cup C| \leq 2\alpha \cdot \text{OPT}(G) \leq (2 - 2/k) \cdot \text{OPT}(G).$$

□

Nous donnons maintenant quelques applications du théorème 17. On rappelle qu'un graphe est k -dégénéré si tout graphe induit possède un sommet de degré $\leq k$ (voir aussi le paragraphe 2.5.2).

- Les arbres sont 1-dégénérés.
- Les graphes planaires-extérieurs sont 2-dégénérés.
- Les graphes planaires sont 5-dégénérés.
- Les graphes de largeur arborescente au plus k sont k -dégénérés.
- Les graphes qui excluent un mineur H à k sommets sont $O(k\sqrt{\log k})$ -dégénérés.

En épluchant un graphe k -dégénéré G par degré minimum, puis en coloriant les sommets dans l'ordre inverse par la première couleur disponible (on parle de l'algorithme FIRSTFIT en Anglais), on peut produire pour G une $(k + 1)$ -coloration en temps polynomial (voir la figure 3.3 pour un exemple). La complexité de cet algorithme est $O(n + m) = O(kn)$ [Pourquoi? Comment?]. Donc, pour les graphes k -dégénérés il existe une $(2 - 2/(k + 1))$ -approximation pour COUVERTURE PAR SOMMETS de complexité $O(kn^{3/2})$.



peut-t-on obtenir 6 couleurs dans un planaire ?

FIGURE 3.3 – Coloration d'un graphe planaire par FIRSTFIT

Pour les graphes planaire, il est possible d'obtenir une 4-coloration en temps polynomial (l'algorithme de 4-coloration n'est pas trivial!). Donc, il existe une $3/2$ -approximation pour COUVERTURE PAR SOMMETS dans les graphes planaires, et c'est le meilleur facteur d'approximation connu pour cette classe là.

Le meilleur facteur d'approximation connu pour COUVERTURE PAR SOMMETS dans le cas général est $2 - \Theta(1/\sqrt{\log n})$ [Kar04].

3.4 COUVERTURE PAR ENSEMBLES

Le problème de la couverture par ensembles (*Set Cover* en Anglais) généralise COUVERTURE PAR SOMMETS. L'intérêt est qu'il capture beaucoup d'autres problèmes, dont ENSEMBLE DOMINANT.

COUVERTURE PAR ENSEMBLES :

Instance: deux ensembles B et \mathcal{F} tels que $\bigcup_{S \in \mathcal{F}} S = B$

Question: trouver $\mathcal{C} \subseteq \mathcal{F}$ de cardinalité minimum qui couvre B , c'est-à-dire tel que $\bigcup_{S \in \mathcal{C}} S = B$.

L'ensemble \mathcal{F} est donc une famille de sous-ensembles de B . On peut toujours supposer que les ensembles de \mathcal{F} sont distincts, puisque dans toute solution, il n'y a pas d'avantage à prendre plusieurs fois le même ensemble. La condition $\bigcup_{S \in \mathcal{F}} S = B$ garantit qu'il existe toujours au moins une couverture \mathcal{C} , en particulier $\mathcal{C} = \mathcal{F}$, et donc une plus petite.

Il existe une version évaluée du problème, encore plus générale, où chaque ensemble $S \in \mathcal{F}$ possède un coût $c(S)$. Il s'agit de trouver une couverture \mathcal{C} de B de coût minimum, le coût de \mathcal{C} étant défini par $c(\mathcal{C}) = \sum_{S \in \mathcal{C}} c(S)$. Cette version ne sera pas discutée dans ce cours, mais tous les résultats se transposent dans ce cadre plus général. Ici on considère que $c(S) = 1$ pour tout $S \in \mathcal{F}$ si bien que $c(\mathcal{C}) = |\mathcal{C}|$ est simplement le nombre d'éléments de \mathcal{C} .

COUVERTURE PAR SOMMETS d'un graphe G est une instance de ce problème avec $B = E(G)$ et $\mathcal{F} = \{E_u : u \in V(G)\}$ où E_u est l'ensemble des arêtes incidentes au sommet u (voir figure 3.4).

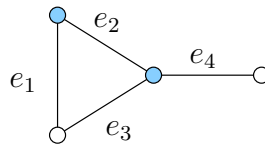


FIGURE 3.4 – COUVERTURE PAR SOMMETS est un cas particulier de COUVERTURE PAR ENSEMBLES. Dans cet exemple, $B = \{e_1, e_2, e_3, e_4\}$ et $\mathcal{F} = \{\{e_1, e_2\}, \{e_2, e_3\}, \{e_2, e_3, e_4\}, \{e_4\}\}$. Et une couverture possible de taille 2 est $\mathcal{C} = \{\{e_1, e_2\}, \{e_2, e_3, e_4\}\}$.

ENSEMBLE DOMINANT d'un graphe G est aussi une instance de ce problème avec $B = V(G)$ et $\mathcal{F} = \{B(u, 1) : u \in V(G)\}$ est l'ensemble des boules de rayon un.

Le problème de COUVERTURE PAR ENSEMBLES est équivalent à un autre problème appelé *Hitting Set* : on veut trouver un ensemble $C \subseteq B$ de taille minimum qui intersectent tous les ensembles de \mathcal{F} . Cela revient à échanger le rôle de B et \mathcal{F} . COUVERTURE PAR ENSEMBLES est aussi le dual (un max au lieu d'un min) du problème *Set Packing* : il s'agit de trouver la plus grande famille $\mathcal{C} \subseteq \mathcal{F}$ d'ensembles deux à deux disjoints.

3.4.1 Algorithme glouton

De façon surprenante, l'algorithme glouton le plus simple et le plus naturel est essentiellement la meilleure approximation qu'on puisse espérer pour ce problème. On ne démontrera pas dans ce cours que tel est bien le cas.

Algorithme SETCOVERGLOUTON(B, \mathcal{F})

1. $R := B$ et $\mathcal{C} := \emptyset$
 2. Tant que $R \neq \emptyset$ faire
 - (a) Sélectionner un ensemble $S \in \mathcal{F}$ tel que $|S \cap R|$ est maximum
 - (b) $\mathcal{C} := \mathcal{C} \cup \{S\}$ et $R := R \setminus S$
 3. Renvoyer \mathcal{C}
-

Dans la pratique, on peut à chaque itération, supprimer S de la famille \mathcal{F} pour éventuellement accélérer la recherche du prochain S . Pour analyser les performances de cet algorithme, on aura besoin de la fonction *harmonic* définie par :

$$H(n) := \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}.$$

On a $\ln n < H(n) \leq 1 + \ln n$ pour tout $n \geq 1$, mais aussi que $\lim_{n \rightarrow \infty} H(n) = \gamma + \ln n$, où $\gamma = 0.5772156649\dots$ est la constante d'Euler-Mascheroni.

Théorème 18 *L'algorithme SETCOVERGLOUTON(B, \mathcal{F}) est une $H(k)$ -approximation pour COUVERTURE PAR ENSEMBLES de taille minimum, où $k = \max_{S \in \mathcal{F}} |S|$.*

Preuve. À chaque boucle de la ligne 2, l'ensemble R représente l'ensemble des éléments restant à couvrir. Comme $\bigcup_{S \in \mathcal{F}} S = B$, il existe donc toujours un ensemble $S \in \mathcal{F}$ couvrant un élément quelconque de R . Donc R diminue strictement à chaque itération. À la fin de l'algorithme, il est clair que les ensembles qui se trouve dans \mathcal{C} couvrent tous les éléments de B . Donc \mathcal{C} est bien une couverture par ensembles.

Il est évident que l'algorithme a une complexité en temps qui est polynomiale en la taille des instances, c'est-à-dire en $|S| + |\mathcal{F}|$. Notez que la taille de \mathcal{F} n'est pas nécessairement proportionnelle à celle de S . Il est même possible que $|\mathcal{F}| = 2^{\Omega(|S|)}$. Il n'empêche que l'algorithme reste de complexité polynomiale.

Il nous reste à montrer que c'est une $H(k)$ -approximation où $k = \max_{S \in \mathcal{F}} |S|$. Soit \mathcal{C}^* une couverture de taille optimale. Posons $k^* = \max_{S \in \mathcal{C}^*} |S|$ la taille maximum d'un ensemble sélectionné pour \mathcal{C}^* . Bien sûr $k^* \leq k$. On va montrer que l'algorithme est en fait d'une $H(k^*)$ -approximation.

Dans ce cours, on va montrer que c'est une $H(n)$ -approximation où $n = |B|$, ce qui est certes moins fort car $k \leq n$, mais plus simple à démontrer. On représente l'instance de

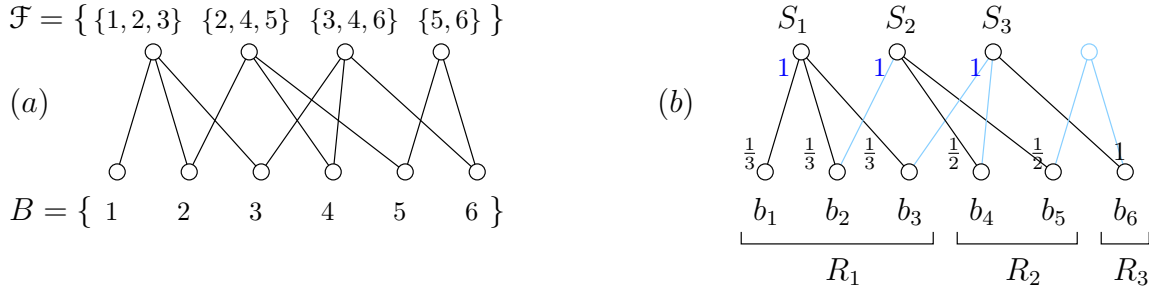


FIGURE 3.5 – (a) Représentation d’une instance (B, \mathcal{F}) de COUVERTURE PAR ENSEMBLES. (b) Coûts (en bleu) et prix lors d’une exécution possible de l’algorithme glouton.

COUVERTURE PAR ENSEMBLES par un graphe biparti. Sur un premier niveau on place les ensembles de \mathcal{F} , et sur le second les éléments de B . On met une arête entre $S \in \mathcal{F}$ et $b \in B$ si $b \in S$ comme illustré sur la figure 3.5(a).

On suppose une exécution de l’algorithme (cf. figure 3.5(b)), et l’on note S_i l’ensemble sélectionné par l’algorithme à l’étape $i \in \{1, \dots, |\mathcal{C}|\}$, \mathcal{C} étant la couverture renvoyée par l’algorithme. On note aussi R_i l’ensemble R au début de l’itération i (donc avant sa mise à jour). On numérote b_1, \dots, b_n les éléments de B dans l’ordre où ils sont couverts par l’algorithme, les *ex æquo* étant classés arbitrairement.

On va évaluer de deux façons différentes le coût de l’algorithme qui est le nombre d’itérations $|\mathcal{C}|$. La première façon est de compter combien de sommets du 1er niveau sont sélectionnés. Chaque ensemble S_i compte donc pour 1. On note¹ $\text{coût}(S_i) = 1$. La seconde est de répartir équitablement, à chaque ensemble sélectionné S_i , le coût de S_i sur les nouveaux éléments qu’il couvre. Si l’élément b_j devient couvert par S_i , il reçoit alors un *prix* égale au coût de S_i divisé par le nombre d’éléments nouvellement couverts par S_i . Formellement,

$$\text{prix}(b_j) = \frac{\text{coût}(S_i)}{|S_i \cap R_i|} = \frac{1}{|S_i \cap R_i|} .$$

En quelque sorte, lorsque l’algorithme sélectionne S_i , il « achète » les $t = |S_i \cap R_i|$ nouveaux éléments au prix unitaire de $1/t$, ce qui lui coûte bien sûr $\text{coût}(S_i) = 1 = t \cdot (1/t)$.

On a donc deux façons d’exprimer le coût de l’algorithme : la somme des coûts des ensembles sélectionnés, ou bien la somme des prix des éléments couverts. Autrement dit,

$$|\mathcal{C}| = \sum_{i=1}^{|\mathcal{C}|} \text{coût}(S_i) = \sum_{j=1}^n \text{prix}(b_j) .$$

Considérons l’itération i . On va supposer que b_j est le prochain élément couvert par S_i .

1. Pour la version valuée du problème, il aurait fallu poser $\text{coût}(S_i) = c(S_i)$.

On remarque que l'algorithme, en choisissant l'ensemble S_i qui maximise $|S_i \cap R_i|$, choisit dans les éléments restant un élément de prix minimum, puisque $\text{prix}(b_j) = 1/|S_i \cap R_i|$.

Pour borner le facteur d'approximation, il faut relier $|\mathcal{C}|$ et $|\mathcal{C}^*|$. Montrons qu'il existe un élément $b^* \in R_i = \{b_j, \dots, b_n\}$ encore non couvert de prix $\leq |\mathcal{C}^*|/|R_i|$. En effet, nous savons que $|\mathcal{C}^*|$ ensembles suffisent pour couvrir B et donc *a fortiori* R_i . Le coût total $|\mathcal{C}^*|$ induit un prix moyen sur les éléments de R_i qui est au plus $|\mathcal{C}^*|/|R_i|$. Il existe donc un élément $b^* \in R_i$ de prix $\leq |\mathcal{C}^*|/|R_i|$. En particulier,

$$\text{prix}(b_j) \leq \text{prix}(b^*) \leq \frac{|\mathcal{C}^*|}{|R_i|} = \frac{|\mathcal{C}^*|}{n - j + 1}.$$

Donc,

$$|\mathcal{C}| = \sum_{j=1}^n \text{prix}(b_j) \leq \sum_{j=1}^n \frac{|\mathcal{C}^*|}{n - j + 1} = |\mathcal{C}^*| \cdot \left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{1} \right) = |\mathcal{C}^*| \cdot H(n).$$

L'algorithme glouton est donc une $H(n)$ -approximation. \square

Comment modifier l'algorithme et son analyse dans le cas valué ? Dans cette version chaque ensemble $S \in \mathcal{F}$ possède un coût $c(S)$, et on cherche à minimiser le coût total de la couverture \mathcal{C} à savoir $\sum_{S \in \mathcal{C}} c(S)$.

D'autres algorithmes un peu meilleurs existent pour résoudre COUVERTURE PAR ENSEMBLES lorsque k est « petit ». On parle du problème *k-Set Cover*. Le facteur d'approximation est $H(k) - 1/2$ pour tout $k \geq 1$, $H(k) - 1/2 - 1/390$ si $k \geq 4$. Et il y a encore d'autres ratios un peu meilleurs, de la forme $H(k) - c_k$, lorsque $k \geq 6$ (voir [ACK09]).

Essayons de construire des instances critiques pour cette approximation. On considère des ensembles S_1, \dots, S_t partitionnant B , chaque S_i avec 2^i éléments, et deux ensembles T_0 et T_1 . L'ensemble T_0 contiennent la moitié de chacun des éléments des S_i , et T_1 l'autre moitié. L'ensemble B comprend un total de $n = \sum_{i=1}^t 2^i = 2^{t+1} - 2$ éléments. Notons que $t = \log_2(n - 2) - 1 \geq \log_2 n - 2$ pour tout $n \geq 2$.

[DESSIN]

On a $|T_0| = |T_1| = n/2 = 2^t - 1$. Donc l'algorithme glouton va choisir S_t qui couvre 2^t éléments. La situation se répète ensuite, puisque les éléments non couverts de T_0 (ou de T_1) sont au nombre de $2^{t-1} - 1$ alors que S_{t-1} en couvre toujours un de plus. Et ainsi de suite. Au final, l'algorithme renvoie une couverture de taille $t \geq \log_2 n - 2$, alors que l'optimale est de taille 2. L'approximation est donc au moins $\frac{1}{2} \log_2 n - 1 \approx 0.72 \ln n$. Dans notre cas, $k = n/2$, et $H(k) \approx \ln n - O(1)$. Ce n'est pas tout à fait une instance critique.

La valeur $H(n)$ n'est pas le meilleur facteur d'approximation pour l'algorithme glouton. Dit autrement, le théorème 18 ne donne pas la meilleure analyse possible de cet algorithme en fonction de $n = |B|$. Il a été démontré dans [Sla96] que l'algorithme glouton avait un facteur d'approximation $< \ln n - \ln \ln n + 0.78$, et il existe des instances où le facteur est

$> \ln n - \ln \ln n + 0.31$. La constante 0.78 est en fait l'approximation de $-1 + \ln 2$, et le 0.31 de $3 + \ln \ln 32 - \ln 32$. Il a été montré [Fei98] qu'aucun algorithme polynomial, sauf si $N=NP$, ne peut donner un facteur d'approximation inférieur à $(1 - o(1)) \ln n$. L'algorithme glouton est donc, d'une certaine façon, quasiment le meilleur possible.

Que donne cet algorithme pour COUVERTURE PAR SOMMETS et ENSEMBLE DOMINANT ?

Pour ENSEMBLE DOMINANT, l'algorithme glouton peut, pour tout entier $k \geq 1$, renvoyer un ensemble dominant de taille $4k$, alors qu'un ensemble de taille $3k$ existe. Il suffit de prendre une collection indépendante de $K_{1,3}$ où chaque arête est remplacée par un chemin de longueur 2 (voir figure 2.6). L'algorithme glouton renverra 4 au lieu de 3. Notons que pour les graphes de degré ≤ 3 , comme ce contre-exemple, l'algorithme glouton fournit une $H(4)$ -approximation, $H(4) = 25/12 \approx 2.08$. Ce graphe n'est donc pas une instance critique pour les graphes de degrés ≤ 3 .

Question : quelle est la pire instance de ENSEMBLE DOMINANT pour l'algorithme glouton ? Notons que l'étoile est aussi un contre-exemple pour COUVERTURE PAR SOMMETS, avec le même facteur d'approximation.

3.4.2 Un second algorithme

Un paramètre important d'une instance (B, \mathcal{F}) de COUVERTURE PAR ENSEMBLES est sa *fréquence*, le nombre maximum d'ensembles de \mathcal{F} contenant un élément donné de B . Dans la représentation sous forme de graphe biparti de l'instance (B, \mathcal{F}) (cf. figure 3.5), cela correspondant au degré maximum d'un sommet de B .

Plus formellement, la fréquence est définie par :

$$f(B, \mathcal{F}) = \max_{b \in B} |\{S \in \mathcal{F} : b \in S\}|.$$

Toute instance de COUVERTURE PAR SOMMETS a une fréquence $f \leq 2$. Chaque arête $(u, v) \in B$ apparaît dans deux ensemble E_u et E_v . Pour ENSEMBLE DOMINANT la fréquence est $f \leq \Delta + 1$, un de plus que le degré maximum du graphe.

L'algorithme suivant est une variante de l'algorithme glouton précédent.

Algorithme SETCOVERFREQ(B, \mathcal{F})

1. $R := B$ et $\mathcal{C} := \emptyset$
 2. Tant qu'il existe $b \in R$ faire
 - (a) $\mathcal{S} := \{S \in \mathcal{F} : b \in S\}$, la famille d'ensembles de \mathcal{F} contenant b
 - (b) $\mathcal{C} := \mathcal{C} \cup \mathcal{S}$ et $R := R \setminus \bigcup_{S \in \mathcal{S}} S$
 3. Renvoyer \mathcal{C}
-

Théorème 19 *L'algorithme SETCOVERFREQ(B, \mathcal{F}) est une f -approximation pour COUVERTURE PAR ENSEMBLES de taille minimum, où $f = f(B, \mathcal{F})$ est la fréquence de l'instance.*

Preuve. Clairement SETCOVERFREQ est polynomial et renvoie une couverture de B . Calculons le facteur d'approximation.

On remarque que les familles \mathcal{S} construites à l'étape 2(a) sont deux à deux disjointes, et de cardinalité au plus f . Ensuite, à chaque fois qu'on ajoute \mathcal{S} à \mathcal{C} à l'étape 2(b), au moins un ensemble $S \in \mathcal{S}$ appartient à la couverture optimale \mathcal{C}^* , sinon l'élément b ne pourrait pas être couvert par \mathcal{C}^* . Il suit que $|\mathcal{C}| \leq f \cdot |\mathcal{C}^*|$. La taille de la couverture retournée par l'algorithme (\mathcal{C}) est au plus f fois celle de l'optimale (\mathcal{C}^*). \square

Instance critique pour cet algorithme ?

Pour ENSEMBLE DOMINANT, $f \leq \Delta$ et donc l'approximation de l'algorithme SETCOVERGROUTON est meilleure car $H(\Delta+1) < \Delta+1$ dès que $\Delta > 1$. Mais pour COUVERTURE PAR SOMMETS, $f = 2$, l'algorithme SETCOVERFREQ donne l'approximation standard.

3.4.3 Algorithme exact

Dans l'algorithme suivant, on suppose connu un algorithme (polynomial???) SETCOVER-DEG2(B, \mathcal{F}) retournant une couverture de taille minimum si chaque élément de \mathcal{F} est de taille ≤ 2 .

Algorithme SETCOVERMIN(B, \mathcal{F})

1. Simplifier \mathcal{F} en posant $\mathcal{F} := \{S \cap B : S \in \mathcal{F}\}$.
 2. Si $|\mathcal{F}| = \emptyset$, renvoyer 0.
 3. Si $\exists S \neq R \in \mathcal{F}$ avec $S \subseteq R$, renvoyer SETCOVER($B, \mathcal{F} \setminus \{S\}$).
 4. Si $\exists v \in B$ et un unique $S \in \mathcal{F}$ avec $v \in S$, renvoyer $1 + \text{SETCOVERMIN}(B \setminus S, \mathcal{F})$.
 5. Choisir $S \in \mathcal{F}$ de plus grande taille.
 6. Si $|S| = 2$, renvoyer SETCOVER-DEG2(B, \mathcal{F}).
 7. Renvoyer $\min \{\text{SETCOVERMIN}(B, \mathcal{F} \setminus \{S\}), 1 + \text{SETCOVERMIN}(B \setminus S, \mathcal{F})\}$.
-

La complexité de cet algorithme est $2^{0.305(|B|+|\mathcal{F}|)} \cdot n^{O(1)}$. En utilisant une technique de mémorisation (et donc un espace exponentiel), on montre que l'algorithme a une complexité de $2^{0.299(|B|+|\mathcal{F}|)} \cdot n^{O(1)}$.

La complexité exacte de cet algorithme n'est pas connue. Elle est au moins $\Omega(3^{n/4}) = \Omega(2^{0.396n})$... pour mds [FGK09] (mds=minimum dominating set).

Bibliographie

- [ACK09] Stavros Athanassopoulos, Ioannis Caragiannis, and Christos Kaklamanis. Analysis of approximation algorithms for k -set cover using factor-revealing linear programs. *Theory of Computing Systems*, 45(3) :555–576, 2009.
- [Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4) :634–652, 1998.
- [FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5) :Article No. 25, August 2009.
- [Kar04] George Karakostas. A better approximation ratio for the Vertex Cover problem. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume Report No. TR04-084, October 2004.
- [Sla96] Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 435–441. ACM Press, May 1996.