Analyse d'Algorithme

Cyril Gavoille

LaBRI
Laboratoire Bordelais de Recherche
en Informatique, Université de Bordeaux

gavoille@labri.fr

11 décembre 2023
– 172 pages –

Master 2

Cours d'introduction à la complexité paramétrique et aux algorithmes d'approximation

Pré-requis : algorithmique ; notions de théorie des graphes

Quelques ouvrages de référence :

- *Kernelization – Theory of Parameterized Preprocessing*
  Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Meirav Zehavi
  Cambridge 2019
- *Invitation to Fixed-Parameter Algorithms*
  Rolf Niedermeier
  Oxford 2006
- *Parameterized Complexity Theory*
  Jörg Flum, Martin Grohe
  Springer 2006
- *Fundamentals of Parameterized Complexity*
  Rodney G. Downey, Michael R. Fellows
  Springer 2013
- *Algorithmes d'approximation*
  Vijay V. Vazirani
  Springer 2006
- *Approximation Algorithms for NP-hard Problems*
  Dorit S. Hochbaum
  PWS 1995
- *Algorithm Design*
  Jon Kleinberg, Éva Tardos
  Addison Wesley 2006
- *Parameterized Algorithms*
  Marek Cygan, Fedor V. Fomin, *et al.*
  Springer 2015
- *Computational Complexity : A Modern Approach (4th edition)*
  Sanjeev Arora, Boaz Barak
  Cambridge 2016

Table des matières

Avant-propos

Ces notes retranscrivent deux cours de Master 2. L'un intitulé « Analyse d'Algorithme » et l'autre « Algorithmique Appliquée » dont les objectifs sont un peu différents. Pour le cours d'Algorithmique Appliquée, le chapitre 1 n'est pas nécessaire avec l'Avant-propos. Pour le cours d'Analyse d'Algorithme peut commencer dès le chapitre 1.

Objectifs

Il y a deux objectifs pour le cours d' Algorithmique Appliquée :

- Balayer quelques techniques/approches algorithmiques pour résoudre des problèmes rencontrés en « Recherche et Développement » (R&D). Analyser ces techniques pour comprendre pourquoi elles marchent, identifier leurs points forts et leurs faiblesses. Car en algorithmique tout est question de compromis : tel algorithme sera efficace pour tel type de données. Et c'est à l'algorithme de s'adapter aux données.
- Programmer ces approches et les tester sur un problème en grandeur réelle. Cela se traduira par un projet, dont quelques uns sont décrits page 13, qui s'étalera pendant la durée du cours soit un semestre.

Exemples d'applications et de problèmes R&D

En règle générale on est confronté à plein de données, beaucoup de contraintes et le problème n'est pas bien défini. Les contraintes principales étant : il faut produire rapidement une solution et que cela marche sur les exemples fournis par le client. Voici quelques exemples, que l'on espère suffisamment bien définis.

Ex1. Reconstruction de code génétique à partir de fragments.

On dispose d'un ensemble de gènes, chacun codé par des mots, disons sur l'alphabet $\{A, C, G, T\}$, et l'on cherche une façon de les ordonner de sorte que i-ème gène dans cet ordre, disons g_i, matche éventuellement avec g_{i+1} sur un segment et que le nombre

total de lettres matchantes des segments entre gènes successifs soit le plus grand possible. On donne aussi des contraintes sur les ordres possibles, comme le fait que deux gènes donnés ne doivent pas se suivre. Autrement dit toutes les permutations (ordres) possibles ne sont pas permises.

Pour une entreprise qui développe des tests ou profils ADN, une technique algorithmique plus efficace que les entreprises concurrentes peut mener à des gains substantiels.

Ex2. Simulateur de vol réaliste (stage M2, 2017).

Une entreprise de jeu vidéo veut développer en partenariat avec Microsoft™ un simulateur de vol basé sur les cartes de la terre, cartes basées sur des images satellites de Bing Maps. Le problème est donc de synthétiser ces cartes de façon à pouvoir déduire les altitudes et de pouvoir changer ainsi à volonté (du joueur et du simulateur) les perspectives, l'éclairage, l'ombrage, la visibilité, etc. Ce travail existe déjà pour les cartes urbaines, mais pas pour les paysages. Les images satellites sont des tuilages *bitmaps* et il convient de détecter les textures (végétation, eau, ...) malgré les ombres. L'extraction des ombres permettent aussi de déduire les altitudes. Le problème est difficile en particulier pour les montagnes enneigées où l'identification des ombres et des rochers n'est pas facile. Il a fallu développer de nombreuses solutions algorithmiques, après une recherche bibliographique sur le sujet, et faire des compromis (vitesse de calcul, qualité du résultat, difficulté à programmer l'algorithme et donc à maintenir le code résultant).

Ex3. Collecte d'ordure.

Il s'agit de produire un logiciel permettant d'optimiser la collecte des ordures incluant de nombreuses contraintes : sens interdits, horaires et jour de marcher, planning des équipes et des tournées, vidage des camions. Bien sûr, utiliser les cartes de l'IGN nécessite des droits et licences que les entreprises ne peuvent pas toutes se payer. Il y a donc un système « maison » de récolte des cartes à base de GPS dans les camions pour la reconstruction des routes. La reconstruction des routes est un problème à part entière.

Après une modélisations non triviale, cela revient à résoudre un problème du Voyageur de commerce (ou une de ces variantes).

Se méfier de l'optimisation du programme en fonction d'un jeu d'essai limité (parfois à un seul exemple). Via des paramètres que l'on optimise au fur et à mesure des essais, on peut, sans s'en apercevoir, coder en dure la solution optimale. On le voit parfois : des constantes arbitraires apparaissent, et elles permettent de meilleures solutions. Si le problème est proche de celui du Voyageur de commerce, on peut par exemple mettre des poids sur les sommets pour que l'algorithme développé privilégie certains points, et ne pas se rendre compte qu'on est en train de coder la solution. C'est aussi le problème du « sur-apprentissage » (*overfitting*) que l'on observe dans les algorithmes d'apprentissages (*machine learning*) où il y a trop de paramètres par rapport aux données.

Ex4. Analyse de code (stage M2, 2019).

Une entreprise Anglaise, qui commercialise des outils pour l'analyse de code, veut implémenter les recommandations du MISRA C:2012. Il s'agit d'un ensemble de règles (norme) pour l'écriture de code en `C` émises par l'association éponyme. Une des règles énonce que « lors d'une déclaration, un tableau ne doit pas être initialisé partiellement. » Dans l'extrait suivant, qui est du `C` parfaitement valide, seules les déclarations des tableaux `A` et `B` respectent *a priori* la règle.

```
int i;
int A[i=3];
int B[] = { ++i, 7 };
int C[5] = { 7, i };
int D[5] = { [ 3 ] = 7 };
int E[5] = { E[3], 7, E[i], E[2], 0[E+1] };
int F[5] = { [ 1? ({1+2;}) : ({main(F[1],NULL); 1?3:3;}) ] = 7 };
```

Dans le tableau `E`, seuls les éléments `E[1]` et `E[4]` seront correctement initialisés à `7`.

Ex5. Classification pour le *e*-commerce.

Les plateformes de *e*-commerce possèdent des millions de références qui évoluent très rapidement et qui doivent être référencées dans leur base de données pour construire le site *web*. Il s'agit alors de pouvoir automatiquement décoder l'étiquetage des fournisseurs vers le système de référence de la plateforme. Détecter par exemple qu'il s'agit de vêtement pour femme, de jupe, détecter la couleur, ou détecter les caractéristiques importantes de taille pour les pneus par exemple.

Pour ce dernier exemple, C-Discount™ à lancer un concours pour trouver le meilleur algorithme de classification. (La classification optimale étant établie à l'avance). Le meilleur algorithme proposé atteignait 75% ce qui très insuffisant. Du 90% est nécessaire, sans quoi le site *web* est jugé trop erroné par le client potentiel.

Le *leader* mondial est Amazon™, en 2015 tout au moins. Et à cette date sa puissance de calcul était de 20 fois supérieure à la somme totale de la puissance de calcul utilisée par tous leurs concurrents, y compris Google™ (partie *e*-commerce seulement). C'est la puissance de calcul, et donc des algorithmes, qui permet d'améliorer significativement la qualité du résultat.

Importance des algorithmes

L'Intelligence Artificielle (IA) a maintenant une définition très large qui inclut beaucoup d'aspects de l'algorithmique. On lit souvent que les robots sont animés par de l'IA (ce qui n'est pas faux). Mais en réalité ce qui jusqu'à présent anime les robots ce sont

bien des algorithmes, et pas des réactions biochimiques se déroulant au sein d'un corps cybernétique.

Dans le grand public, le mot « algorithme » a maintenant presque remplacé le mot « logiciel », et le terme « IA » est de plus en plus souvent utilisé pour dire « algorithme ».

On parlera peut-être d'IA lorsque la tâche réalisée est complexe (prise de décision comme pour les jeux d'échecs ou de go), et/ou qui s'appuie sur de très grandes bases de données (toutes les parties jamais jouées). Pour beaucoup, une forme d'intelligence émerge lorsque la réponse résulte d'une exécution difficilement traçable, après avoir augmenté sa base de connaissances avec une période d'auto-apprentissage [1]. Alors qu'un algorithme bien compris, comme ceux que l'on trouvent dans une calculatrice pour le calcul de $\sin(x)$, sera jugé moins intelligent. Mais juger l'intelligence d'une réponse proportionnellement à sa non-explicabilité c'est sans doute faire preuve d'irrationalité ou de croyances mystiques.

Ajoutons que les algorithmes d'apprentissage profond (*deep learning*) sont une classe d'algorithmes dont les nombreux paramètres (les poids synaptiques du réseau de neurones) peuvent être optimisés en fonction de données et de réponses attendues. Ces méthodes se révèlent très efficaces pour certains problèmes, car il existe un algorithme efficace d'apprentissage (optimisation des paramètres par rétro-propagation). Il faut cependant bien avoir conscience qu'il s'agit d'optimiser une fonction (les paramètres d'un algorithme) parmi un ensemble grand mais nécessairement limité. Un peu comme si on cherchait un algorithme d'une forme spéciale : un algorithme utilisant un nombre borné de variables, deux boucles `for` et un test `if` par exemple. Si cela peut marcher dans de nombreux cas intéressants, c'est absolument inefficace pour certains. La fonction $x \mapsto \sin(x)$ par exemple semble difficile à apprendre à un réseau profond de neurones, même en se limitant à la double précision de la norme IEEE 754, soit 80 bits comme le type `double` en `C`. Le réseau résultant risque d'être au mieux particulièrement inefficace, au pire incorrect. Idem pour la multiplication de matrices. Combien de triplets de matrices $\{0,1\}^{n^2} \times \{0,1\}^{n^2} \to \{0,1\}^{n^2}$ l'algorithme devra-t'il apprendre avant d'être juste sur les 2^{2n^2} paires de matrices booléennes $n \times n$ possibles ?

Il est aussi bien connu que, pour chaque réseau de neurones optimisé pour la reconnaissance d'images (après apprentissage donc), il est possible de construire un bruit universel U ayant la propriété suivante : chaque image I classée c par le réseau sera classée $c' \neq c$ une fois bruitée par U. De façon remarquable, le flou ajouté est quasiment imperceptible pour l'oeil humain (voir la figure 1). Ce bruitage fait donc échouer

1. Comme cela a été réalisé pour le jeu d'échec et de go où l'IA AlphaZero a pu atteindre un niveau inégalé et surhumain seulement à partir des règles du jeu et de temps (de quelques heures à quelques jours) pour l'apprentissage et la simulation de millions de parties. Cependant, c'est bien l'algorithme de parcours de l'arbre de jeu qui fait l'intelligence de cette IA. Ici l'apprentissage sert au choix des branches à évaluer. Voir également l'article [MKT+22] sur l'explicabilité. À noter qu'une simple modification des règles (par exemple, un coup sur dix, un pion est supprimé) ne permet plus à la machine de jouer sans un ré-apprentissage complet, et donnera très facilement l'avantage à l'humain.

chaque image apprise par le réseau ... Cela pose de sérieux problème de sécurité (pour la reconnaissance de visages pour l'accès à une zone protégée par exemple). Il n'y a aucune parade (c'est un théorème !) sinon de dégrader la qualité de l'image en la compressant, ce qui réduit le bruit mais d'autant les performances de discrimination de l'algorithme.

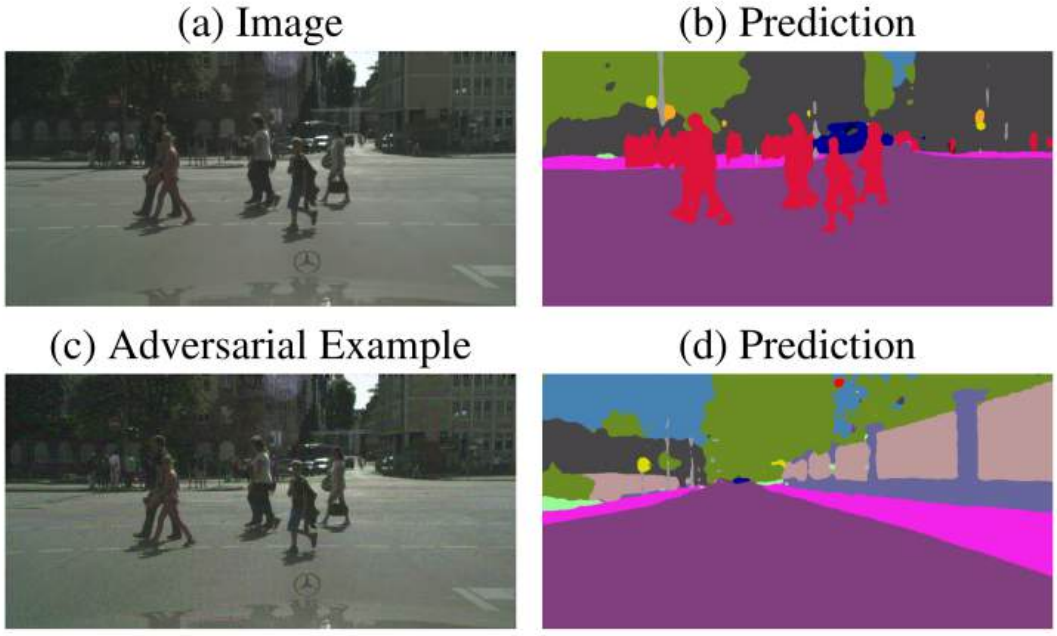


FIGURE 1 – Exemple d'images bruitées issues de [MMBF17]. (a) Image de ville et ses zones de prédiction (b). (c) Image perturbée avec un bruit universel et ses nouvelles zones de prédiction (d). D'après les auteurs, la prédiction est similaire pour les autres images du jeu d'essais lorsqu'elles sont perturbées avec le même bruit.

Des travaux ont même essayé de minimiser le nombre de pixels à changer pour faire échouer le réseau neuronal. Et l'on arrive souvent à un très petit nombre, voir un seul pixel. On parle de *One Pixel Attack*, cf. [SVS19]. Le pixel dépend évidemment de l'image d'entrée, ce n'est plus une attaque universelle.

Notons aussi que les réponses des algorithmes d'apprentissage sont nécessairement entâchés d'erreurs, mais doivent être distingués de celles données par les algorithmes d'approximation. Pour ces derniers, la qualité de la réponse est garantie alors que pour les algorithmes d'apprentissage la ganrantie est au mieux statistique, tout comme dans un algorithme probabiliste. Aussi, un réseau de neurones avec un taux de réussite de 99% n'est pas forcément meilleure qu'un humain avec un taux de 90%. Cela est du aux attaques malicieuses humaines (ou non). Il pourrait être facile de systématiquement exploiter les failles (=1%) du réseau de neurones (disons en changeant un seul pixel...), alors qu'exploiter les 10% d'erreurs humaines le soit moins. Et cet argument ne s'applique pas qu'aux réseaux de neurones, mais à n'importe quelle machine déterministe.

Notons que, contrairement à un algorithme probabiliste, un réseau de neurones, dont les poids ont étés fixés par apprentissage, produit une réponse déterministe (mêmes entrées, mêmes sorties).

Pour terminer sur les algorithmes d'apprentissage et les réseaux de neurones, citons les travaux de [LKNW15] qui ont développé un « système d'apprentissage » simple (basé sur le renforcement positif) permettant la détection de tumeur cancereuse du sein avec un taux de réussite de 72% (et même 90% sur les images tests), soit aussi bien que des radiologues expérimentés humains, mais surtout à un coût bien moindre.

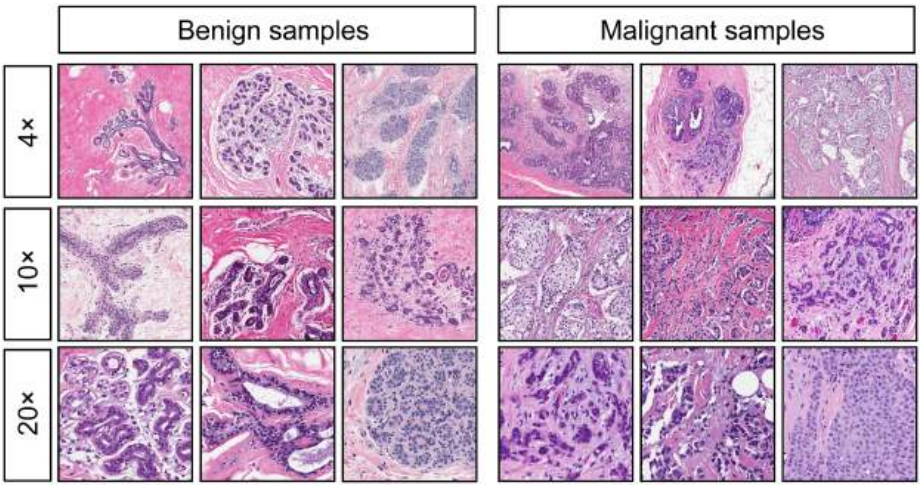


Figure 2 – Tumeurs bénignes ou cancéreuses du sein. Illustration extraite de [LKNW15].

Pourtant, le réseau de neurones en question n'était qu'un simple cerveau biologique (voir figure 3). On ne parlera pas plus des algorithmes d'apprentissage.

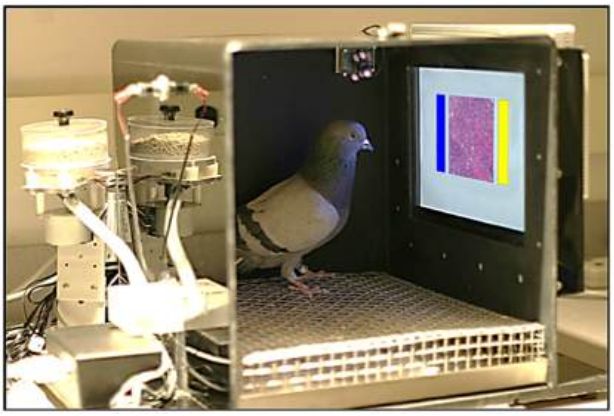


Figure 3 – C'est le cerveau de pigeons qui permet un apprentissage surhumain pour la détection des tumeurs, cf. [LKNW15] et aussi la vidéo.

Il est donc important d'avoir une culture « algorithmique » à défaut d'avoir une

expertise. Il est essentiel de connaître la limites de ces algorithmes. Beaucoup d'éléments d'information sont disponibles sur le *web*. Faut-il encore être capable de savoir quoi chercher. Comme trouver une implémentation efficace des fonctions de hachage lorsqu'on ne connaît pas le concept de fonction de hachage ? Il faut pour cela de la culture informatique et/ou algorithmique, être curieux et faire en sorte, à la sortie de vos études, de ne pas être en concurrence directe avec indiens et chinois qui étaient plus de 2.7 milliards en 2020 (1.380 + 1.402) et dont une fraction non nulle savent très bien écrire du code et à moindre coût (et pendant que vous dormez à cause du décalage horaire). En passant, il est prévisible que les ordinateurs écrivent bientôt eux-mêmes le code. Et qui va concevoir et paramétrer ces méta-algorithmes ?

Pour illustrer l'importance de la curiosité (disons algorithmique), citons deux exemples réels d'embauche.

Ex1. Une entreprise bordelaise (en 2015) a proposé à un candidat à l'embauche (un M2 Info) de résoudre le problème suivant. Dans un fichier contenant $n = 10\,000$ points du plan, trouver les points qui forment des carrés (s'il y en a). Il est demandé de fournir le résultat la semaine suivante, tout en expliquant la méthode utilisée. Déterminer si 4 points donnés forment un carré est facile, encore faut-il se souvenir des formules du déterminant et donnant la distance (le carré suffit) entre deux points. Notez bien qu'un algorithme en n^4 donnerait $(10^4)^4 = 10^{16} = 10^7 \times 10^9 = 10^7$ secondes (10 millions de secondes donc). Sur un ordinateur cadencé à 1 GHz, cela fait 2 777 heures ou 115 jours environ. On peut faire en fait $n^2 \cdot \log n$, soit $10^8 \cdot \log_2(10^4) = 1.32$ secondes sur le même ordinateur. [*Aide*. Quand on fixe la diagonale d'un carré, où se trouvent les quatre points potentiels du carré ?] [*Exercice*. Peut-on faire mieux ? Combien de carrés peut-on former avec n points du plan ? [2]]

La question était relativement gentille comparée à celle où le fichier aurait contenu 100 000 points (en effet, 10,000 points ce n'est jamais que le nombre de points d'une grille 100×100). La méthode naïve aurait donné $(10n)^4 = 10\,000 \times n^4$, soit 3 170 années, à comparer aux 2 minutes 45 pour l'autre.

Ex2. Entretien d'embauche chez Google™. Il faut déterminer le point de rupture à la chute d'une balle fabriquée en série. Le seul moyen que l'on dispose est de lâcher la balle d'un certain étage et de voir si elle résiste ou elle se casse. Il y a n étages possibles. La question est de savoir combien de tests unitaires sont nécessaires pour déterminer le point de rupture si l'on dispose d'une seul balle ? Si l'on dispose de 2 balles ? Question plus difficile (et non posée au candidat) : si l'on dispose d'assez de balles, quel est le nombre minimum de tests qu'il faut réaliser ?

2. Pour aller plus loin, on pourra consulter [vKdB91], qui donne par exemple un algorithme pour décider s'il existe un carré aligné sur les axes – ce qui est un problème différent – en temps $O(n\sqrt{n \log n}) = O(n^{3/2+\varepsilon})$. Bizarrement, un algorithme légèrement plus rapide en $O(n^{3/2})$ est donné dans le cas de la détection d'un rectangle aligné sur les axes [vKdB91, th. 5].

D'autres exemples d'embauches chez Google™ peuvent être visionnés sur le *web* [3].

De l'application à l'algorithme

Il y a une interaction forte entre applications, algorithmes et logiciels. Schématiquement on retrouve (cf. figure 4) :

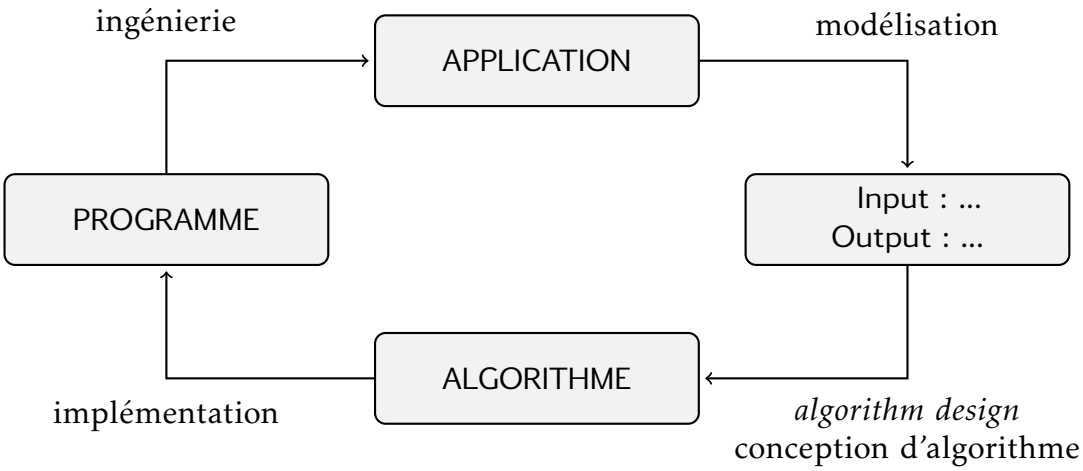


Figure 4 – Schéma illustrant le passage de l'application à l'algorithme.

Le passage APPLICATION → ALGORITHME, soit les deux premiers arcs, est le plus difficile à réaliser et c'est là que nous intervenons (surtout le 2e arc – *algorithm design*). Les deux suivantes sont plutôt de l'ordre du Génie Logiciel. Savoir s'il est utile de programmer un algorithme donné est l'objectif premier de l'Analyse d'Algorithme. Avant tout développement, il est bon de savoir si l'algorithme donnera une réponse avant la fin de l'Univers. Certes, si l'algorithme est particulièrement simple, on pourrait le programmer et « voir » ce qu'il en est. Sauf qu'on risque de devoir le stopper avant sa terminaison (et la fin de l'Univers) sans rien en déduire. Notons que, même pour des *Input* données, une petite variation sur l'*Output* à souvent un impact considérable sur la complexité et l'efficacité de l'algorithme. C'est le cas par exemple pour le problème des carrés parmi n points du plan évoqué page 7 : savoir s'il existe au moins un carré ou les lister tous sont deux problèmes différents de complexité différente, $O(n^{3/2+\varepsilon})$ *vs.* $\Omega(n^2)$.

Il arrive que le 2e arc en bas à droite n'existe même pas. Il est bon de se rappeler qu'il existe des problèmes pour lesquels aucun algorithme ne peut, même en théorie, en venir à bout. Et cela même avec une complexité arbitrairement grande. Une façon simpliste de le voir est de se représenter un algorithme comme la méthode calculatoire qu'une machine à d'associer une *Output* à chaque *Input*. Si on restreint les entrées/sorties à seulement des entiers naturels, l'algorithme est donc ni plus ni moins la représentation

3. *How to: Work at Google — Example Coding/Engineering Interview*.

	0	1	2	3	4	5	6	…
f_0	5	2	0	5	5	0	2	…
f_1	6	9	7	2	8	2	1	…
f_2	8	3	1	0	5	2	4	…
f_3	7	0	0	4	7	4	0	…
f_4	1	2	1	2	1	2	1	…
f_5	3	6	2	5	1	6	9	…
…	…	…	…	…	…	…	…	…

TABLE 1 – Diagonalisation de Cantor. On peut représenter une fonction entière par un vecteur de ses valeurs possibles. Si les fonctions pouvaient être ordonnées comme ci-dessus, où se situerait la fonction définie par 6 0 2 5 2 7…, c'est-à-dire les éléments diagonaux +1 modulo 10 ?

d'une certaine fonction $f : \mathbb{N} \to \mathbb{N}$. Un argument simple de diagonalisation (dite de Cantor, cf. la table 1) montre que les fonctions de $\mathbb{N} \to \mathbb{N}$ ne peuvent pas être énumérées, alors que les algorithmes (en fait leurs codes) eux le peuvent (selon l'ordre lexicographique par exemple). Dit autrement, il y a bien plus de fonctions que d'algorithmes, mêmes si dans les deux cas leurs nombres est infini. Et donc certaines fonctions n'ont tout simplement pas d'algorithme.

Quelles techniques algorithmiques ?

Dans ce cours nous verrons :

1. Algorithmes exacts
   - FPT, réduction de données
   - décomposition arborescente
   - programmation dynamique
2. Algorithmes approchés

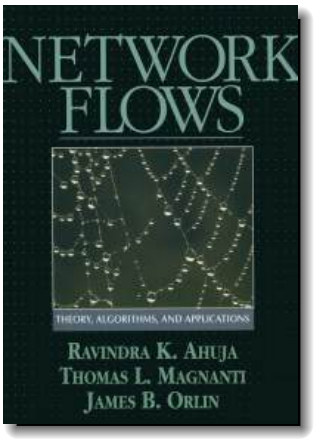
Et les algorithmes que nous ne verrons pas (ou peu) :

- Algorithmes probabilistes ;
- Algorithmes de recuit simulé ;
- Algorithmes génétiques ;
- Algorithmes d'apprentissage ;
- Algorithmes quantiques ;
- …

En fait, il n'y a pas tant d'algorithmes que cela permettant de résoudre les problèmes

les plus courants de manière exacte et en temps polynomial. On peut en distinguer quatre types :

- Les algorithmes de parcours et de recherche linéaire : on veut par exemple la somme des valeurs d'une base de données. Il s'agit essentiellement d'algorithme de complexité linéaire, soit $O(n)$.
- Les algorithmes qui effectuent une réorganisation des données, comme le tri ou le pré-calcul de certaines structures de données. Leur complexité est souvent [4] en $O(n \log n)$.
- Les algorithmes basés sur des calculs de distance entre paires de points, comme le calcul de plus courts chemins dans un graphe. Ils résultent bien souvent de produit matriciel qui est en $O(n^\omega)$ pour un certain $\omega \in [2,3[$.
- Les algorithmes de calcul de flot dans un graphe valué (capacités et coûts sur les arcs), comme le calcul d'un couplage maximum, c'est-à-dire trouver un maximum d'arêtes indépendantes. La forme la plus générale s'appelle *Minimum Cost b-Flow* ou b-flot de coût minimum. L'algorithme a pour complexité $O(n^{2.5})$ avec un algorithme complexe [5].



Le problème du b-flot de coût minimum est défini comme suit [6]. On a un graphe orienté, et chaque arc uv a une capacité $c(uv)$ ainsi qu'un coût $\omega(uv)$. De plus chaque sommet u possède une demande (ou budget) $b(u)$ telle que $\sum_u b(u) = 0$. Il s'agit d'abord de trouver un b-flot f, soit une fonction qui associe à chaque arc uv une valeur $f(uv) \leqslant c(uv)$ respectant les demandes ou bien de dire qu'il n'en existe pas. Plus précisément, respecter les demandes signifie que pour chaque sommet u :

$$b(u) = \sum_{uv^+} f(uv^+) - \sum_{uv^-} f(uv^-)$$

où uv^+ est un arc sortant de u et uv^- un arc entrant. Lorsque $b(u) = 0$, on retrouve la loi de conservation, $b(u) > 0$ pour une source et $b(u) < 0$ pour un puits. Ensuite, le b-flot doit être de coût minimum, c'est-à-dire tel que :

$$\text{coût}(f) := \sum_{uv} f(uv) \cdot \omega(uv) \quad \text{soit minimum.}$$

4. Même si cela semble proche, la différence entre parcourir et réorganiser des données peut être importante en pratique. Chercher une voiture sur un parking avec une plaque d'immatriculation donnée et très différent de réorganiser les voitures en les garant par ordre croissant de plaque.

5. À sa découverte, qui a été une surprise, on a cru qu'on allait pouvoir résoudre de nombreux problèmes jugés jusqu'ici difficiles.

6. Des livres entiers ont été consacré à ce problème et aux différentes méthodes de résolutions, comme [AMO93]. Le livre [KV10, chp. 8 à 10] est aussi un excellent point de départ.

Beaucoup de problèmes peuvent être résolus en se ramenant à l'un de ces algorithmes ou alors à une combinaison de ceux-là. Quelques exemples rapides :

- Les problèmes du calcul du flot maximum et des plus courts chemins peuvent être résolu à l'aide d'un algorithme de b-flot de coût minimum. Voir Wikipédia pour les réductions [7].
- Le problème *Vector Racer* (cf. figure 5) sur une grille $n \times n$ se ramène au calcul d'un plus court chemin dans un graphe de configurations de taille polynomiale. Même réduction pour le jeu de la Tour de Hanoï, avec n disques et p piquets (un livre récent de 500 pages y a été consacré [HKP18]).
- Le problème (bien réel) des greffes de reins (cf. figure 6) se ramène à celui du couplage maximum dont les algorithmes polynomiaux sont très proches de celui d'un calcul de flot maximum (cf. le paragraphe 3.2.1).
- Le problème du couplage maximum dans un graphe biparti se calcule à l'aide du calcul d'un flot maximum. Bien que pour le cas général il n'existe pas de réduction du couplage maximum depuis celui du flot maximum, les techniques de résolutions sont très proches.

Au-delà, la plupart des problèmes sont bien souvent NP-complets (ou NP-difficiles), ce qui veut dire qu'on en connaît pas de solution algorithmique polynomiale.

Le fameux problème P=NP met en lumière le fait qu'il est particulièrement difficile de montrer qu'il n'existe pas, pour un problème donné, un algorithme de complexité relativement faible. Si on se sait pas dire grand chose sur de tels problèmes (qu'on suspecte être de complexité non-polynomiale), on ne connaît pas non plus la complexité de certains problèmes polynomiaux pourtant simples, comme celui du produit de matrices $n \times n$. On sait faire en [8] $O(n^\omega)$ où $\omega < 2.3729$ mais rien ne dit qu'on peut pas faire $O(n^{2.1})$, voir $n^2 \text{polylog}(n)$ ou $O(n^2)$, soit un temps linéaire en la taille de l'entrée ! En algorithmique des graphes par exemple, on dit souvent qu' « à la fin ça se termine par un algorithme linéaire ». Il n'empêche qu'on ne sait toujours pas déterminer si un graphe possède un triangle en temps linéaire, soit $O(n^2)$ dans le cas dense. (Le calcul de la matrice d'adjacence au carré indique s'il existe un chemin de longueur deux entre chaque paire (u, v) de sommets. On peut ainsi vérifier s'il y a ou pas un triangle pour chaque arête uv du graphe. Cela donne un algorithme qui, en temps $O(n^2)$, se réduit au produit de matrices et donc de complexité $O(n^\omega)$, ce qui pour le problème du triangle dans un graphe général est la meilleure complexité connue. Pour les graphes peu denses, on peut faire en $O(m\sqrt{m})$, où m est le nombre d'arêtes, ce qui est mieux si $m = o(n^{2\omega/3}) < n^{1.59}$.

7. Par exemple, on peut rajouter deux sommets s et t connectés à tous les autres. Puis en fixant suffisamment bien les capacités de ces nouveaux arcs, on peut calculer un b-flot de coût minimum avec un algorithme de flot maximum.

8. La constante ω est l'exposant dans la complexité du produit de matrices $n \times n$, valeur qui est régulièrement améliorée. En 2014, on avait $\omega < 2.3729$ [LG14][VW14].

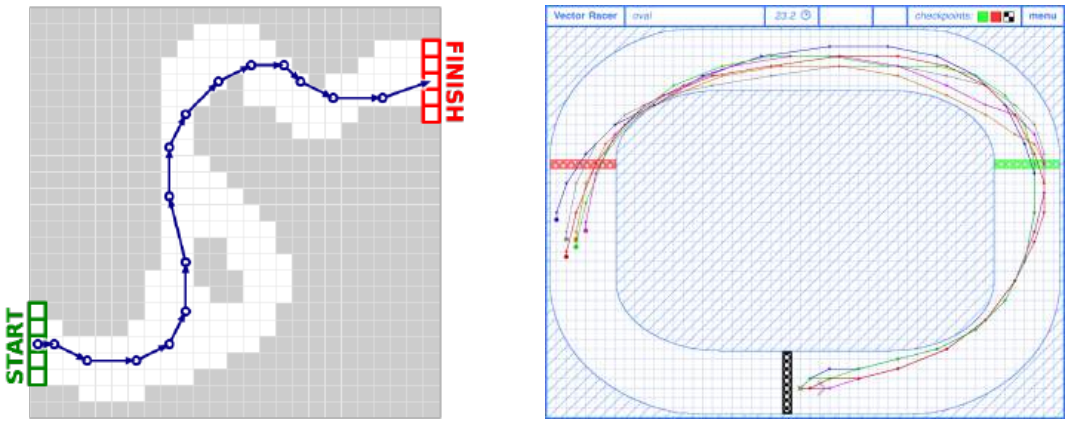


FIGURE 5 – Il s'agit d'une course où l'on se déplace selon un vecteur vitesse dont chaque composante peut être, à chaque étape, incrémenter, décrémenter ou conserver. Il s'agit de minimiser le nombre d'étapes (et donc le temps) pour rejoindre l'arrivée tout en restant sur la piste. On peut généraliser le problème où l'arrivée est remplacée par un ensemble de *check-points* qui doivent être visités en un minimum de temps, soit une version temporelle du VOYAGEUR DE COMMERCE (voir [CRS20]). On peut obtenir le temps optimal en calculant un plus court chemin dans le graphe de configurations : chaque sommet est un quadruplet $(x, y, \partial x, \partial y)$ où (x, y) est une position valide et $(\partial x, \partial y)$ une vitesse ayant menée à cette position. Un arc entre deux sommets indique que l'on peut de déplacer de l'un à l'autre en une étape. On peut montrer que ce graphe possède seulement $O(n^3)$ sommets et arêtes. [*Question*. Pourquoi ? Quel est son degré sortant ?]. Image empruntée de [Eri19] et du cite harmmade.com.

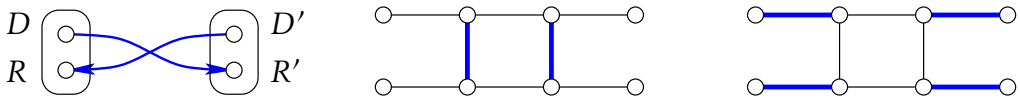


FIGURE 6 – Aux USA en 2014, il y avait 100 000 personnes en attente d'une greffe de reins, dont 5 000 par an en décèdent. On propose aux paires donneurs-receveurs (D, R) non-compatibles un échange avec une paire donneur-receveur (D', R') permettant une greffe croisée : D donne un rein à R' et D' le donne à R. On modélise le problème par un graphe où les sommets sont toutes les paires donneurs-receveurs en mettant une arête entre toute paire rendant possible les greffes croisées. Comme un donneur ne peut être impliqué que dans une seule greffe, il s'agit alors de choisir un couplage du graphe, si possible de taille maximum. La généralisation à des échanges triangulaires ne permet pas de calculer une solution en temps polynomial (sauf si P=NP). Voir l'extrait de leçon inaugurale de Claire Mathieu au Collège de France.

Il est également connu que le nombre de triangles vaut exactement $\mathrm{tr}(A^3)/6$ où A est la matrice d'adjacence du graphe et $\mathrm{tr}(M)$ la trace de la matrice M, c'est-à-dire la somme de ses éléments diagonaux.)

Bien sûr tout cela est empirique puisque la théorie de la complexité dit que pour chaque complexité $C(n)$, il existe un problème de taille n et de complexité $C(n)$ sans être de complexité inférieure. Malheureusement, pour chacun de ces problèmes, on ne sait pas déterminer ceux qui sont dans NP de ceux qui n'y sont pas.

Quels problèmes, quels projets ?

- En 2016, le projet était de déterminer le score maximal réalisable dans une poignée de Uno, c'est-à-dire déterminer quelles cartes et dans quel ordre les poser pour maximiser le score (version du jeu seul donc). Après réduction, cela revient à trouver le chemin le plus long dans un graphe. On met une arête entre deux cartes si elles peuvent être jouées à la suite (même couleur ou même numéro).
- En 2017, le projet consiste à déterminer, par exemple, le positionnement des quatre fantômes du jeu de Pac-Man de sorte à pouvoir capturer le Pac-Man le plus rapidement possible quel que soit son emplacement de départ.
- En 2018, année de la 2e étoile des Bleus, le projet est un problème de placement de robots défenseurs devant les buts en vue d'un match de la RoboCup.
- En 2019, on se remet sur le terrain de foot pour une variante du projet 2018, histoire de fêter la 4e victoire d'affilée de l'équipe du LaBRI à la RoboCup dans la catégorie Humanoïde *KidSize* qui a été remportée à Sydney en juillet 2019, avec la première touche lancée de l'histoire réalisée en finale. Cela sera l'occasion de préparer l'édition 2020 qui se déroulera en juin à domicile au Parc des Expositions à Bordeaux Lac.
- En 2020, c'est encore une variante du projet autour de la RoboCup qui a été proposé. Tout comme le changement de date des Jeux Olympiques de Tokyo, elle se déroulera finalement en 2021, et pas en 2020, et le tout dans une version distancielle à Bordeaux. Elle sera en présentielle en juillet 2023.
- En 2021, crise sanitaire oblige, le projet aura pour objet le placement des centres de vaccinations en région parisienne, avec plusieurs contraintes.
- En 2022, année des incendies géants de Gironde cet été là, il s'agit de sécuriser des villages par des zones de pare-feu qui doivent être placer optimalement.
- En 2023, transition énergétique oblige (même si l'équipe du LaBRI a gagné pour la 5e fois la RobotCup à domicile), le projet aura pour objectif de satisfaire une demande du parlement européen, à savoir de mettre à disposition une station de recharge rapide pour véhicule électrique tous les 60 km d'autoroute.

FIGURE 7 – Quelques projets des années passées.

Bibliographie

[AMO93] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Network Flows : Theory, Algorithms, and Applications*, Prentice Hall, 1993. ISBN : 9780136175490, 013617549X.

[CRS20] A. CASTEIGTS, M. RAFFINOT, AND J. SCHOETERS, *VectorTSP : A traveling salesperson problem with racetrack-like acceleration constraints*, Tech. Rep. 2006.03666v2 [cs.DS], arXiv, August 2020.

[Eri19] J. ERICKSON, *Algorithms*, Creative Commons, 2019. http://jeffe.cs.illinois.edu/teaching/algorithms/.

[HKP18] A. M. HINZ, S. KLAVŽAR, AND C. PETR, *The Tower of Hanoi – Myths and Maths (2nd edition)*, Birkhäuser, 2018.

[KV10] B. Korte and J. Vygen, *Optimisation combinatoire : Théorie et algorithmes*, Springer, 2010. ISBN : 978-2-287-99036-6, 978-2-287-99037-3. DOI : 10.1007/978-2-287-99037-3.

[LG14] F. Le Gall, *Powers of tensors and fast matrix multiplication*, in 39th International Symposium on Symbolic and Algebraic Computation (ISSAC), ACM Press, July 2014, pp. 296–303. DOI : 10.1145/2608628.2608664.

[LKNW15] R. M. Levenson, E. A. Krupinski, V. M. Navarro, and E. A. Wasserman, *Pigeons (columba livia) as trainable observers of pathology and radiology breast cancer images*, Public Library of Science (PLoS) ONE, 10 (2015), pp. 1–21. DOI : 10.1371/journal.pone.0141357.

[MKT+22] T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik, *Acquisition of chess knowledge in alphazero*, Tech. Rep. 2111.09259v3 [cs.AI], arXiv, August 2022.

[MMBF17] J. H. Metzen, C. K. Mummadi, T. Brox, and V. Fischer, *Universal adversarial perturbations against semantic image segmentation*, Tech. Rep. 1704.05712v3 [stat.ML], arXiv, July 2017.

[SVS19] J. Su, D. V. Vargas, and K. Sakurai, *One pixel attack for fooling deep neural networks*, IEEE Transactions on Evolutionary Computation, 23 (2019), pp. 828–841. DOI : 10.1109/TEVC.2019.2890858.

[vKdB91] M. J. van Kreveld and M. T. de Berg, *Finding squares and rectangles in sets of points*, BIT Numerical Mathematics, 31 (1991), pp. 202–219. DOI : 10.1007/BF01931281.

[VW14] V. Vassilevska Williams, *Multiplying matrices faster than Coppersmith-Winograd*, January 2014. http://theory.stanford.edu/~virgi/matrixmultf.pdf.

CHAPITRE

1 Introduction

> *Rien ne sert de penser, il faut réfléchir avant.*
>
> — Pierre Dac

Sommaire

1.1 À quoi sert l'analyse d'algorithme ?

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, non conforme, ou qui ne termine pas). 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. En quelque sorte donner une preuve constructive.

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 difficiles 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. Poser $i := 3$
3. Tant que $2i \leqslant n$ faire :
   - si i et $n - i$ sont deux nombres premiers, renvoyer Vrai, sinon $i := i + 2$
4. Renvoyer Faux.

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

- 3 et 27 sont premiers ? NON ;
- 5 et 25 sont premiers ? NON ;
- 7 et 23 sont premiers ? OUI, et donc il renvoie Vrai.

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

Est-ce que :

$$\forall n > 2 \text{ pair}, \quad \text{Goldbach}(n) = \text{Vrai} \ ?$$

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 $n < 4.1 \times 10^{18}$ (2014). Une preuve (qui reste à confirmer) a été annoncée d'une version plus faible de la Conjecture de Goldbach (voir [Hel14]) : tout nombre impair $n > 5$ est la somme de trois nombres premiers.

Ex2 :

Algorithme Syracuse(n)

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 « $3x + 1$ » traité en détails dans l'ouvrage [Lag10]. C'est vrai pour tout entier $n \leqslant 20 \times 2^{58}$, vérifié par Tomás Oliveira e Silva en 2009 [Lag10, p. 189], ce qui est supérieure à cinq 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] :

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

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 enigmes mathématiques. On pourrait par exemple écrire un programme très compact permettant de vérifier la conjecture de Goldbach :

$$\text{« } n := 4 \text{ ; tant que Goldbach}(n) \text{ faire } n := n + 2 \text{ »}$$

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

1.2 Quelques algorithmes que nous analyserons

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

Ceux que nous n'analyserons pas (ou très peu) :

- Algorithmes probabilistes ;

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.

- Algorithmes de recuit simulé ;
- Algorithmes génétiques ;
- Algorithmes d'apprentissage ;
- Algorithmes quantiques ;
- ...

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 », typiquement NP-complet s'il s'agit d'un problème de décision.

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 appartient à 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 quelque sorte, 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 $\leqslant \sqrt{n}$, 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 $\lceil \log n \rceil$ bits [2]. 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 poly-logarithmique en n. La complexité de l'algorithme du à [AKS04] 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 [3] à $k\log^2 k = O(\log n \cdot (\log\log n)^2)$.

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

```
Input:  integer n > 1.
1.  If (n = a^b for a ∈ 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 ≤ r, output COMPOSITE.
4.  If n ≤ r, output PRIME.[1]
5.  For a = 1 to ⌊√φ(r) log n⌋ do
        if ((X + a)^n ≠ X^n + a (mod X^r − 1, n)), output COMPOSITE;
6.  Output PRIME.
```

FIGURE 1.1 – Test de primalité original extrait de [AKS04, p. 784]. Les auteurs démontrent qu'à l'étape 2, $r \leqslant \lceil \log n \rceil^5$ si bien que le test de l'étape 4 n'a d'intérêt que si $n \leqslant 5\,690\,034$ (dans l'article c'est l'objet de la note de bas de page ligne 4). L'entier $\mathrm{o}_r(n)$ représente le plus petit entier k tel que $n^k = 1 \pmod r$. 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 $\sqrt{\phi(r)} \leqslant \sqrt{r} \leqslant \lceil \log n \rceil^{5/2}$.

En fait, la complexité réelle de l'algorithme est probablement beaucoup moins. Si la conjecture de Sophie Germain est vraie [4], 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 a été proposé avec la même complexité mais dont l'analyse ne repose sur aucune conjecture.

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 $\log^* k = \min\{i : \log^{(i)} n \leqslant 1\}$, cf. [Für09][HvdHL16].

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.

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 objects 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 tout en préservant les distances à un facteur α près. On peut voir un *spanner* comme une sorte de squelette d'un graphe.

Définition 1.1 (spanner) *Un sous-graphe H d'un graphe G est un α-spanner si pour toute paire de sommets $x, y \in V(G)$, $d_H(x,y) \leqslant \alpha \cdot d_G(x,y)$.*

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 coté, 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 1.1 *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 notation 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 $\mathrm{BFS}_G(u,H)$ (*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 $\mathrm{BFS}(u,H)$.

On considère l'algorithme suivant :

Algorithme 3Spanner(G)

Entrée: un graphe G
Sortie: un *spanner* H

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

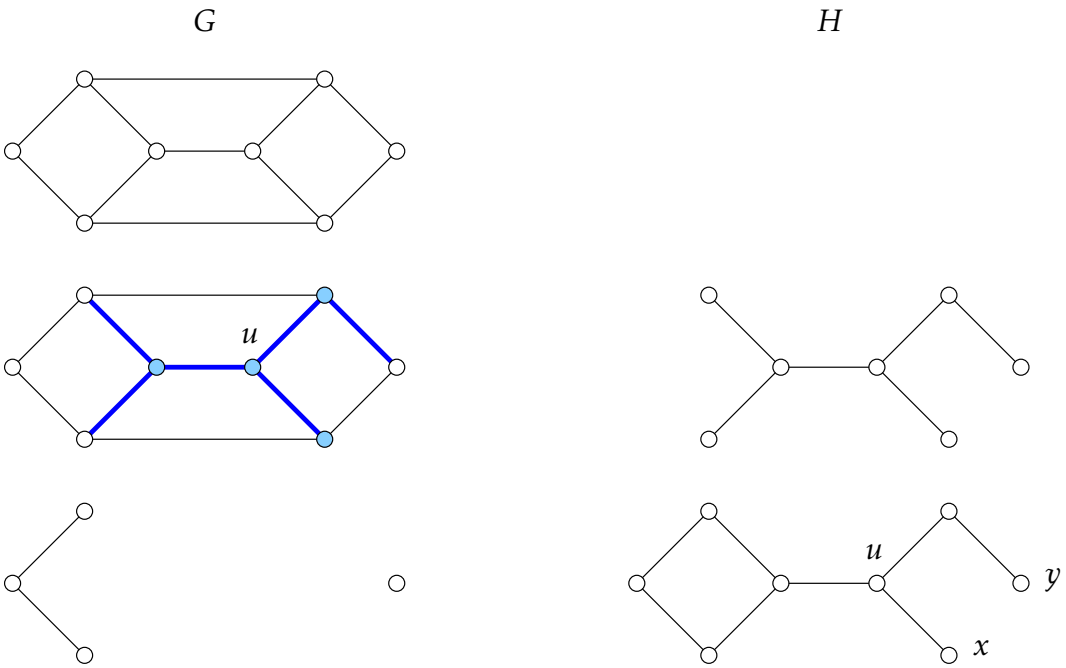


FIGURE 1.2 – Exemple d'exécution de l'algorithme 3Spanner(G) avec $n = 8$ et $\sqrt{n} \approx 2.82$. L'exécution se lit en ligne de haut en bas. Au départ (ligne 1), H est vide. Le sommet u est sélectionné (ligne 2) et un arbre BFS de profondeur 2 est construit puis ajouté à H. Enfin, u est supprimé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

On va d'abord montré qu'entre toute paire de sommets adjacents dans G, il existe dans H un chemin de longueur $\leqslant 3$. 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 $\leqslant 3$, car si chaque arête de G est étirée par un facteur $\leqslant 3$, alors tout chemin de longueur d (en particulier un plus court chemin) sera également étiré par un facteur $\leqslant 3$.

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 ajouter 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 $\leqslant 3$ entre x et y, c'est-à-dire $d_H(x,y) \leqslant 3$.

Montrons que la taille de H est au plus $n^{3/2} - n/2$. Soit $t \geqslant 0$ 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 $n - t \cdot (\lceil \sqrt{n} \rceil + 1) \leqslant n - t\sqrt{n} - t$. On a aussi bien évidemment $\deg_G(u) < \sqrt{n}$ pour les sommets u restant dans G. Le nombre d'arêtes de G à l'étape 3 est donc :

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

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 $t \leqslant n/(\sqrt{n}+1) < \sqrt{n}$. 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 1.1 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 quand 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.

Le théorème 1.1 devient également faux si la taille du 3-spanner est $o(n^{3/2})$, comme le montre la proposition suivante.

Proposition 1.1 *Pour une infinité d'entiers n, il existe un graphe (biparti) à n sommets et plus de $(n/2)^{3/2}$ arêtes sans cycle de longueur inférieur à six.*

En effet, dans un tel graphe il n'est pas possible d'enlever une seule de ses arêtes sous peine d'augmenter la distance entre deux sommets u, v voisins de 1 à 5, car le plus petit cycle passant par uv est de longueur $\geqslant 6$. Dit autrement, si le facteur d'étirement est $s < 5$ (et $\leqslant 3$ en particulier), alors tout s-spanner est de taille $\Omega(n^{3/2})$ pour au moins ce graphe là.

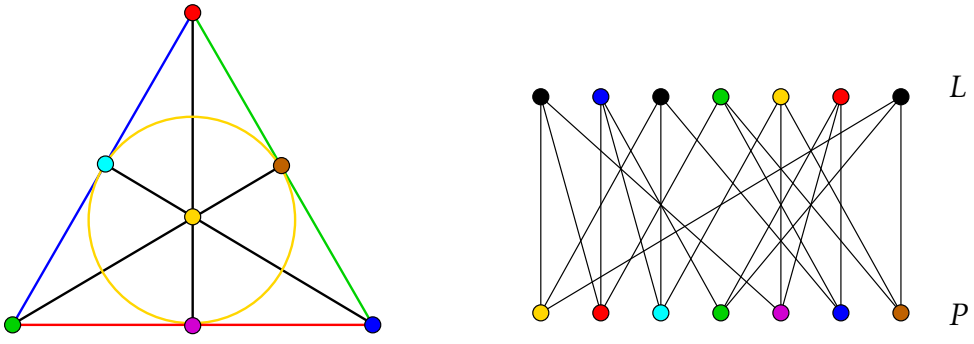
Preuve. La preuve est fortement inspirée de la construction de [PS89], utilisée pour montrer qu'une certaine classe de graphes, les graphes triangulés (c'est-à-dire sans cycle induit de longueur > 3), ne possède pas de 2-spanner de taille $o(n^{3/2})$ arêtes.

La construction est basée sur un plan projectif fini. Un plan projectif est un ensemble de points et de lignes vérifiant les axiomes suivants (cf. Wikipédia) :

(A1) Par deux points passe une seule ligne.
(A2) Deux lignes s'intersectent en exactement un point.
(A3) Chaque ligne passe par au moins trois points.
(A4) Il existe au moins trois points non alignés.

C'est donc comme les points et les droites du plan euclidien sauf que pour ce dernier deux droites s'intersectent soit en un point soit en aucun si elles sont parallèles. Ici l'axiome (A2) implique qu'il n'y a pas de lignes « parallèles ». Donc le plan projectif est un peu comme le plan euclidien où l'on aurait ajouté un point virtuel à chaque direction si bien que toutes les droites parallèles s'y rejoignent, l'ensemble de ces points virtuels formant une ligne d'horizon.

Le plus petit plan projectif fini est composé de 7 points et 7 lignes qui peuvent être dessinés comme sur la figure 1.3.



(a) (b)

FIGURE 1.3 – (a) Plan projectif d'ordre $q = 2$ avec $q^2 + q + 1 = 7$ points et lignes (plan de Fano). Ici le cercle est considéré comme une ligne du plan projectif. (b) Le graphe d'incidence G_q pour $q = 2$. Il s'agit du graphe de Heawood.

Pour chaque plan projectif fini, c'est-à-dire avec un nombre fini de points et de lignes, il existe un entier q appelé « ordre » tel que chaque ligne contient exactement

$q+1$ points et chaque point appartient à exactement $q+1$ lignes. On peut montrer que pour toute puissance q d'un nombre premier [5], il existe un plan projectif (P, L) d'ordre q.

Il peut être construit, pour q premier, à partir des q^2 points $(i, j) \in \{0, \ldots, q-1\}^2$ d'une grille plus $q+1$ points correspondant aux coefficients directeurs de la forme $(1, i)$, pour tout $i \in \{0, \ldots, q-1\}$ et aussi à $(0, 1)$. Cela fait donc $q^2 + (q+1)$ points. À partir de chacun de ces $q+1$ coefficients directeurs on peut construire q droites discrètes modulo q de même direction qui vont partitionner les q^2 points de la grille, q étant premier. Ces droites étant deux à deux parallèles, elles passent aussi par le point correspond à leur coefficient directeur. Une dernière droite reliant seulement les $q+1$ points des coefficients directeurs peut être ajoutée, ce qui porte le total de lignes à $q \cdot (q+1) + 1$. Il suit qu'un plan projectif d'ordre q possède $q^2 + q + 1$ points et autant de lignes.

Pour beaucoup d'ordre q donnés (qui n'est donc pas forcément une puissance d'un premier), on ne sait pas dire s'il existe un plan projectif de cet ordre. Ce n'est pas possible si $q = 1$ ou $2 \bmod 4$ sans que q ne soit la somme de deux carrés. C'est ainsi qu'il n'y a pas de plan projectif d'ordre $q = 6$. Mais pour $q = 12$ par exemple on ne sait pas. Certains jeux de sociétés sont directement inspirés de ces constructions (voir figure 1.4).

FIGURE 1.4 – Le jeu de cartes Dobble™ est basé sur un plan projectif d'ordre $q = 7$. Les points sont les $q^2 + q + 1 = 57$ figures possibles et les lignes sont les 55 cartes du jeu (il en manque deux !), deux cartes ayant exactement une et une seule figure en commun.

Soit q un nombre premier, et (P, L) un plan projectif d'ordre q. On construit le graphe G_q biparti d'incidence de (P, L). Plus précisément [6] :

- $V(G_q) := P \cup L$; et

5. C'est-à-dire que $q = p^t$ pour un certain nombre premier p et entier $t > 0$.

6. On note abusivement « $p \in \ell$ » pour dire que le « point p appartient à la ligne ℓ », alors que les axiomes ne définissent pas ℓ comme un ensemble de points. Ils définissent les propriétés de la relation d'incidence entre « lignes » et « points ».

- $E(G_q) := \{(p,\ell) : \ell \in L, p \in \ell\}$.

Absence de cycle de longueur <6. Comme G_q est biparti, il ne contient pas de cycle de longueur impaire, ce qui exclut les cycles de longueur 3 et 5. Reste les cycles de longueur 4. Supposons qu'il en contienne un. Alors il existe $p, p' \in P$ et $\ell, \ell' \in L$ avec $p, p' \in \ell$ et $p, p' \in \ell'$. Cela implique $|\ell \cap \ell'| \geqslant 2$, ce qui est contradictoire avec l'axiome (A2).

Nombre d'arêtes. D'après la construction on a $|P| = |L| = q^2+q+1 < (q+1)^2$. Le nombre de sommets de G_q est $n = |P| + |L| < 2(q+1)^2$. En particulier, $q+1 > \sqrt{n/2}$. Le nombre d'arêtes de G_q est

$$\begin{aligned} |E(G_q)| &= (q+1)\cdot|P| \\ &> (q+1)\cdot(q+1)^2 = (q+1)^3 \\ &> \left(\sqrt{n/2}\right)^3 = (n/2)^{3/2} \end{aligned}$$

ce qui termine la preuve. □

Remarque. Il est possible d'avoir une borne pour tout nombre de sommets n, et pas seulement pour ceux de la forme $n_q = 2\cdot(q^2+q+1)$ où q est une nombre premier. Pour chaque n, on peut prendre le plus grand nombre premier q tel que $n \geqslant n_q$, utiliser la construction de la proposition 1.1, et de compléter le graphe par $n - n_q$ sommets isolés. Le postulat de Bertrand [7] garantit qu'il existe, pour tout entier $x > 1$, un nombre premier $q \in]x, 2x[$. En choisissant un nombre premier q tel que $\sqrt{n/4} < q+1 \leqslant \sqrt{n/2}$, on obtient, pour un graphe à n sommets sans cycle < 6 ayant un nombre d'arêtes d'au moins $(\sqrt{n/4})\cdot(n/2) > n^{3/2}/4$.

En fait le résultat précédent se généralise de la manière suivante. On retrouve, à $1.5n$ arêtes près, le théorème 1.1 en faisant $k = 2$.

Théorème 1.2 *Soit $k \geqslant 1$ un entier. Tout graphe à n sommets possède un $(2k-1)$-spanner avec moins de $n^{1+1/k} + n$ arêtes.*

Preuve. On considère l'algorithme suivant :

Algorithme $\mathsf{Spanner}_k(G)$

Entrée: un graphe G et un entier $k \geqslant 1$
Sortie: un *spanner* H

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

FIGURE 1.5 – 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 $\leqslant 2k-1$. Soit xy une arête de G. Si $xy \in E(H)$, alors on a un chemin de longueur $1 \leqslant 2k-1$. Supposons $xy \notin E(H)$. Dans l'algorithme, lorsqu'on décide de ne pas ajouter xy à H, on a $d_H(x,y) \leqslant 2k-1$. C'est encore vrai dans le graphe H final. Donc il y a bien dans H un chemin de longueur $\leqslant 2k-1$ entre x et y. L'étirement de H est $\leqslant 2k-1$.

Avant de la démontrer, on va admettre la propriété importante suivante (on peut aussi se référer à [AHL02]) :

Proposition 1.2 *Soient H un graphe à n sommets et $k \geqslant 1$ un entier. Si H n'a pas de cycle de longueur $\leqslant 2k$, alors H possède $< n^{1+1/k} + n$ arêtes.*

Elle est équivalente à dire que si un graphe à n sommets possède $\geqslant n^{1+1/k} + n$ arêtes alors il possède un cycle de longueur $\leqslant 2k$.

Cette proposition permet de conclure quant à la taille de H, car il ne possède pas de cycle de longueur $\leqslant 2k$. 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 $\leqslant 2k$.

Il reste à démontrer la proposition. [*Cyril*. Peut-on faire mieux que $n^{1+1/k}+n$? disons d'un facteur 1/2 en utilisant les cycles $2k+1$ et en rendant biparti le graphe ?]

Preuve de la proposition 1.2. Soit $d = m/n$ où m est le nombre d'arêtes de H. Il s'agit de la densité moyenne, d n'étant pas forcément un entier. L'idée est de montrer que H contient un arbre complet d'arité $\geqslant d$ et de profondeur k, et donc avec au moins $(d-1)^k$ feuilles. Du coup, on obtient n'inégalité $(d-1)^k = (m/n-1)^k < n$, ce qui implique notre résultat.

7. Démontré par Tchebychev en 1850.

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

Algorithme Densifie(H)

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

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

On va montrer que $\overline{H}$ 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 $\overline{H}$ n'a pas de sommet, 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 $\overline{H}$ possède au moins un [8] sommet, disons r.

Soit T un arbre en largeur d'abord (BFS) de racine r couvrant la composante de $\overline{H}$ contenant r. Soit uv une arête de $\overline{H} \setminus T$ telle que u, v sont tous les deux dans T. On note respectivement p_u, p_v leur profondeur dans T. Le point clef est que p_u et p_v sont $\geqslant k$.

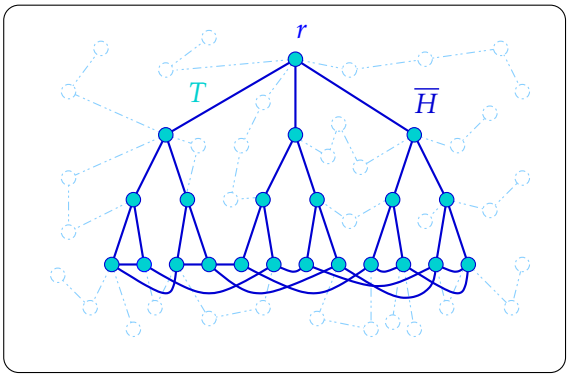


FIGURE 1.6 – Illustration de la preuve de la proposition 1.2. La partie dense $\overline{H}$ de H, ici pour un paramètre de densité $d \in]2, 3]$, contient un arbre complet T de profondeur $k = 3$.

En effet, si $p_u < k$, alors $p_v < k + 1$ puisque T est un BFS. Et donc on pourrait former un cycle $r \to u - v \to r$ de longueur $\leqslant p_u + 1 + p_v \leqslant (k-1) + 1 + (k) = 2k$: impossible ! $\overline{H}$ n'a pas de cycle de longueur $\leqslant 2k$.

Ainsi, le sous-graphe de $\overline{H}$ induit par les sommets à distance $< k$ de r est un arbre. Comme tous les sommets ont un degré $\geqslant d$ dans $\overline{H}$, ils ont aussi un degré $\geqslant d$ dans T s'ils sont à un niveau $< k$. Chaque niveau $i < k$ dans T comprend donc au moins $(d-1)^i$ fils distincts, le "-1" venant du père.

8. En fait, il en possède même au moins $\lceil d \rceil + 1$ puisque ce sommet est de degré au moins $\lceil d \rceil$ dans $\overline{H}$.

Ainsi, la racine r possède $\geqslant d$ fils, chacun ayant $\geqslant d-1$ autres fils distincts, ..., et ceux à la profondeur $k-1$ ayant aussi $\geqslant d-1$ fils. Le nombre de sommets de $\overline{H}$ vérifie donc :

$$\begin{aligned} n \geqslant |V(\overline{H})| &\geqslant 1 + d + d\cdot(d-1) + d\cdot(d-1)^2 + \cdots + d\cdot(d-1)^{k-1} \\ &> d\cdot(d-1)^{k-1} > (d-1)^k = \left(\frac{m}{n} - 1\right)^k . \end{aligned}$$

Donc $(m/n-1)^k < n$, ce qui implique $m < n^{1+1/k} + n$ et termine la preuve de la proposition 1.2. □
Ceci termine la preuve du théorème 1.2. □

En fait, le résultat est valable même si H possède une arête-valuation. Dans ce cas il faut lister les arêtes de H par coût croissant. On remarquera qu'en posant $k = +\infty$, l'algorithme $\mathsf{Spanner}_{+\infty}$ 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).

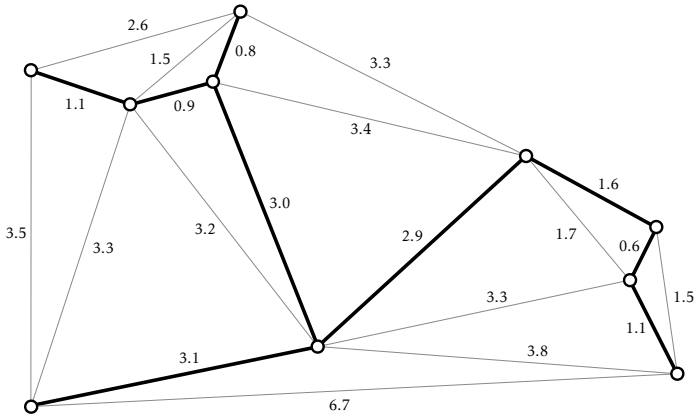


FIGURE 1.7 – Exemple d'arbre couvrant de poids minimum, la valuation correspondant approximativement à la longueur des arêtes.

On pourrait se demander pourquoi on a présenté l'algorithme 3Spanner alors que $\mathsf{Spanner}_k$ fait la même chose pour $k = 2$. En fait, contrairement au théorème 1.2, le résultat du théorème 1.1 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 1.3 ci-dessous.

On utilise la définition suivante :

Définition 1.2 ((α,β)-spanner) *Un sous-graphe H d'un graphe G est un (α,β)-spanner si pour toute paire de sommets $x,y \in V(G)$, $d_H(x,y) \leqslant \alpha\cdot d_G(x,y) + \beta$.*

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 1.3 ci-dessous est donc plus « fort » que le théorème 1.1, puisqu'il l'implique : un $(1, 2)$-*spanner* est un 3-*spanner*.

Théorème 1.3 *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 1.1. (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, $d_H(x, y) \leqslant d_G(x, u) + d_G(u, y)$. Soit v un sommet de $B_G(u, 1) \cap P$. On a $d_G(x, u) \leqslant d_P(x, v) + 1$. De même, $d_G(u, y) \leqslant 1 + d_P(v, y)$. Donc, $d_H(x, y) \leqslant d_P(x, v) + d_P(v, y) + 2$. 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 dans G_0. Donc, $d_H(x, y) \leqslant d + 2 = d_{G_0}(x, y) + 2$ ce qui montre que H est un $(1, 2)$-*spanner*. □

Notons ici que les arêtes doivent avoir un poids uniformes. Sinon, l'étirement additif est $\leqslant 2\Delta$, 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 1.2. Cependant il existe pour tout graphe à n sommets un $(1, 4)$-spanner de $n^{7/5} \cdot \text{polylog}(n)$ arêtes [Che13], et aussi un $(1, 6)$-*spanner* de $O(n^{4/3})$ arêtes [BKMP05][Pet07]. Il a aussi été montré que l'histoire des spanners additifs s'arrête avec l'exposant 4/3 :

Théorème 1.4 ([AB16]) *Pour tout $\varepsilon > 0$, il existe $\delta > 0$ et un graphe à n sommets où tout $(1, \beta)$-spanner de taille $O(n^{4/3-\varepsilon})$ nécessite $\beta \geqslant n^{\delta}$.*

Le résultat est en fait plus général. Tout codage d'un graphe à n sommets permettant de reconstruire les distances à un facteur additif près de n^{δ} nécessite $\Omega(n^{4/3-\varepsilon})$ bits.

Bibliographie

[AB16] A. Abboud and G. Bodwin, *The 4/3 additive spanner exponent is tight*, in 48th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, June 2016, pp. 351–361. doi : 10.1145/2897518.2897555.

[AHL02] N. Alon, S. Hoory, and N. Linial, *The Moore bound for irregular graphs*, Graphs and Combinatorics, 18 (2002), pp. 53–57. doi : 10.1007/s003730200002.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Annals of Mathematics, 160 (2004), pp. 781–793. doi : 10.4007/annals.2004.160.781.

[BKMP05] S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie, *New constructions of (α, β)-spanners and purely additive spanners*, in 16th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2005, pp. 672–681.

[Che13] S. Chechik, *New additive spanners*, in 24th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2013, pp. 498–512. doi : 10.1137/1.9781611973105.36.

[Für09] M. Fürer, *Faster multiplication algorithm*, SIAM Journal on Computing, 39 (2009), pp. 979–1005. doi : 10.1137/070711761.

[Hel14] H. A. Helfgott, *Major arcs for Goldbach's problem*, Tech. Rep. 1305.2897v4 [math.NT], arXiv, April 2014.

[HvdHL16] D. Harvey, J. van der Hoeven, and G. Lecerf, *Even faster integer multiplication*, Journal of Complexity, 36 (2016), pp. 1–30. doi : 10.1016/j.jco.2016.03.001.

[Lag10] J. C. Lagarias, *The Ultimate Challenge : The $3x+1$ Problem*, American Mathematical Society, 2010. isbn : 978-0-8218-4940-8, 978-1-4704-1813-7.

[Pet07] S. Pettie, *Low distortion spanners*, in 34th International Colloquium on Automata, Languages and Programming (ICALP), vol. 4596 of Lecture Notes in Computer Science, Springer, July 2007, pp. 78–89. doi : 10.1007/978-3-540-73420-8_9.

[PS89] D. Peleg and A. A. Schäffer, *Graph spanners*, Journal of Graph Theory, 13 (1989), pp. 99–116. doi : 10.1002/jgt.3190130114.

CHAPITRE

2 Algorithmes exacts

> *C'est à l'algorithmique de s'adapter à l'accroissement des données.*

Sommaire

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 :

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

Il faut donc voir la notion « O » non pas comme une valeur, ou fonction précise, mais plutôt comme un majorant, de surcroît asymptotique. Par exemple, on peut écrire que

$\sin n = O(1)$, ou encore que la complexité d'un programme est en $\log^{O(1)} n$ pour dire qu'elle est *poly-logarithmique* (polynôme en $\log n$). 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 $\sin n = \Omega(1)$, ce qui ne veut évidemment pas dire que $\sin n$ est une fonction constante ou convergente.

Il faut lire cette notation de gauche à droite seulement, et bien se méfier des multiples « O » qui peuvent apparaître dans des expressions. Par exemple, $O(A) = O(B)$ n'est pas pareil que $O(B) = O(A)$, car ici chaque occurrence de la notation « O » se réfère à des constantes n_0 et c *a priori* différentes. Se référer à [Gav23, Chapitre 1] pour un complément et une liste de pièges à éviter.

Pour illustrer les complexités polynomiales et exponentielles, considérons une instance de taille $n = 64$ (comme par exemple une grille 8×8), un ordinateur capable d'exécuter 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	4 μs	1.5^n	3 min
n^4	16 ms	1.7^n	6 jours
n^6	69 s	2^n	585 années

Une complexité en 2.6^n donnerait un temps de 11.5 milliards d'années, une durée proche de l'age de l'Univers.

Il y a d'autres complexités, qui ne sont ni polynomiales ni exponentielles. Par exemple $n^{\log n + O(1)} = n^{O(1)} \cdot 2^{\log^2 n}$ qui la meilleure complexité connue pour résoudre l'isomorphisme de groupes à n éléments, algorithme dû à Tarjan [1].

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 rapide s'exécutera en « seulement » 58 ans et 6 mois au lieu de 585 années.

Pour fixer les idées, 2^n est la complexité de la solution naïve au problème bien connus consistant à vouloir placer le moins de reines possibles sur un échiquier de façon à protéger toutes les cases. Chacune des $n = 8 \times 8$ cases pouvant posséder ou pas de reine, cela

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

fait naïvement 2^n configurations à tester [2]. Notons que le meilleur algorithme résolvant le Voyageur de commerce a une complexité en $2^n \cdot O(n^2)$.

Cependant, si l'on double la taille du problème, l'instance est disons un graphe à $n = 128$ sommets ($\approx$ grille 11×11), alors, même avec le nouvel ordinateur, le temps sera de :

$$T = 2^{2n} = (2^n)^2 \approx (58.5)^2 \text{ années} = 3\,422 \text{ années}$$

et si l'on veut résoudre un problème d'un ordre de grandeur plus important, soit 10 fois plus grand comme un graphe à 640 sommets ($\approx$ grille 25×25) :

$$T = 2^{10n} = (2^n)^{10} \approx (58.5)^{10} \text{ années} = 469 \text{ millions de milliards d'années}$$

ce qui n'a aucun sens. C'est malheureusement l'algorithmique qui doit s'adapter à l'accroissement des données.

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, et les figures 2.2, 2.3 et 2.4 pour des applications) :

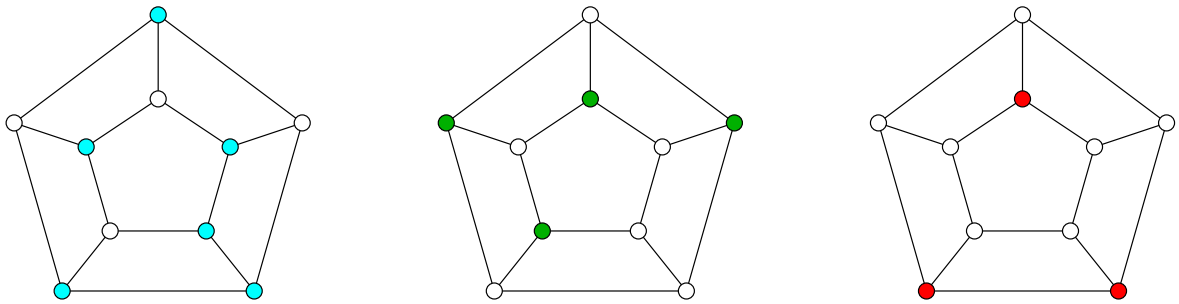
- Couverture par sommets (*Vertex Cover*) : ensemble de sommets contenant au moins une extrémité de chacune des arêtes. En général, on cherche une couverture par sommets de la plus petite taille possible.

- 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.

2. En fait, ne pouvant placer qu'au plus $\sqrt{n}$ reines sur un échiquier $n \times n$, il n'y a que $\sum_{i=0}^{\sqrt{n}} \binom{n}{i} < n^{\sqrt{n}}$ configurations à tester, ce qui est moins que 2^n. [*Question*. Pourquoi ?] Il y a cependant une autre méthode très efficace en pratique : un algorithme probabiliste à rejet. À chaque étape, on choisit une position libre aléatoire parmi les restantes, on place une reine et on recommence jusqu'à toutes les placer. Si à une étape il n'y a plus de place disponible, on efface tout et on tente un nouvel essai. En moyenne, pour une échiquier 8×8, 56 essais suffisent !



Vertex Cover *Independent Set* *Dominating Set*

FIGURE 2.1 – Trois problèmes jouets. La couverture par sommets est minimum car chaque cycle de longueur cinq doit contenir au moins trois sommets. On en déduit que l'ensemble indépendant est maximum. L'ensemble dominant est minimum, un sommet dominant au plus quatre sommets.

Ces trois problèmes sont NP-complets, même pour les graphes planaires. Ils sont mêmes difficiles à approximer pour les graphes cubiques [AK97], c'est-à-dire les graphes où tous les sommets sont de degrés trois comme celui de la figure 2.1. On parle de problèmes *APX-difficiles*. 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 ou un ensemble indépendant maximal est toujours un ensemble dominant, le contraire étant faux en général. [*Exercice.* Pourquoi ?] Également, le complémentaire d'une couverture par sommets est un ensemble indépendant, et qu'inversement, le complémentaire d'un ensemble indépendant est une couverture par sommets. [*Exercice.* Pourquoi ?] Bien sûr, ENSEMBLE INDÉPENDANT est équivalant au problème CLIQUE dans le graphe complémentaire.

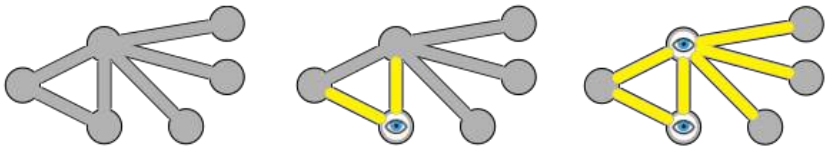


FIGURE 2.2 – Application simple de la recherche d'une couverture par sommets de taille minimum : surveiller les couloirs d'un bâtiment avec le moins de caméras possibles.

Ces problèmes se déclinent en un nombre considérables de variantes et d'extensions. Les plus courantes sont les versions *valuées* des problèmes. La choix de chaque sommet pour appartenir à une solution (couverture par sommets, ensemble indépendant ou dominant) a un coût positif (on parle aussi de *poids* ou de *longueur* suivant le contexte) et on désire un ensemble solution de coût total minimum pour les problèmes de minimisation (comme couverture par sommets et ensemble dominant) ou de coût total maximum pour les problèmes de maximisation (comme ensemble indépendant).

La plupart des algorithmes se généralisent aux versions valuées.

On retrouve la version classique en fixant un coût est unitaire pour chaque sommet. Dans ce cas le coût de la solution correspond à la cardinalité de l'ensemble. Notons qu'en affectant un coût nul ou infini à certains sommets, on peut forcer ou interdire le choix de ces sommets.

On rencontre parfois des variantes où l'ensemble de sommets solution doit en plus satisfaire certaines caractéristiques. Par exemple, pour Ensemble dominant, il est classique d'imposer que le sous-graphe induit par les sommets dominant soit connexe [GK98][RZ11][EJM12], ou encore former un ensemble indépendant [LPL15][GH13] (deux sommets dominants ne devant pas être voisins). Sans être tout à fait les mêmes, les algorithmes sont souvent très inspirés des cas classiques.

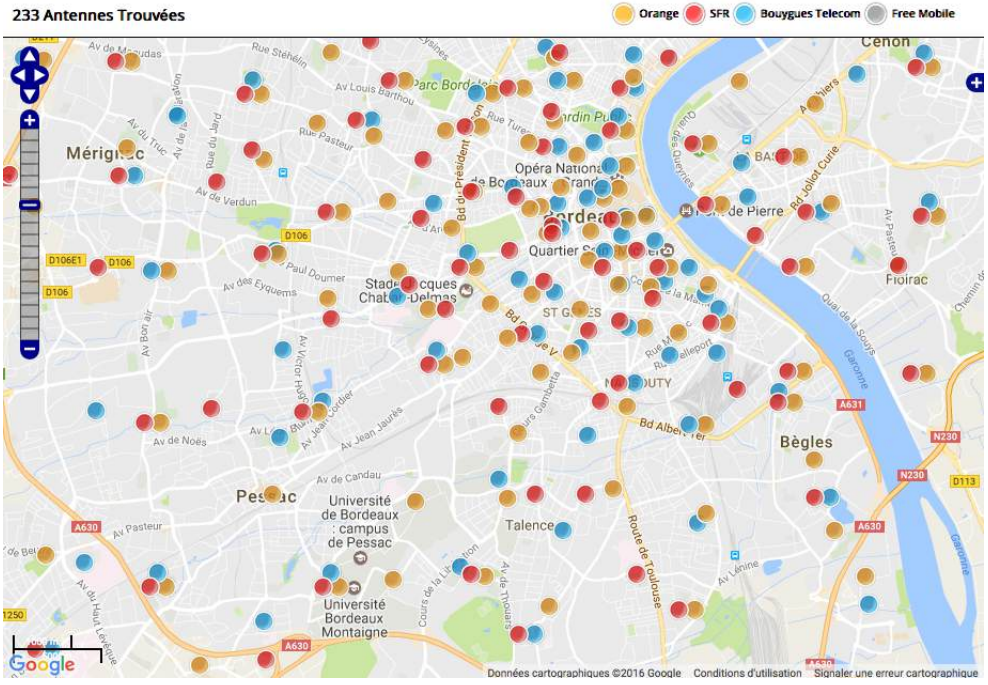


Figure 2.3 – Couverture d'un territoire par un nombre minimum d'antennes radio : chaque point doit (ou devrait) être dans le « voisinage » d'au moins une antenne. Cela revient à calculer un ensemble dominant de taille minimum. © www.antennesmobiles.fr & www.cartoradio.fr.

2.3 Algorithmes exhaustifs (*brute-force*)

Il s'agit d'une technique consistant à lister toutes les possibilités et à vérifier les solutions valides ou à sélectionner les meilleures.

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

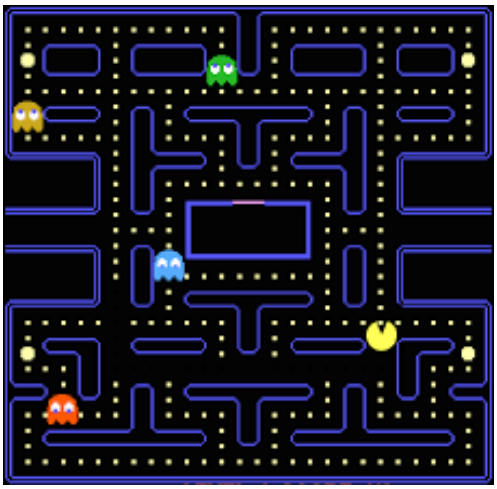



FIGURE 2.4 – Comment placer les quatre fantômes de façon à attraper le Pac-Man le plus rapidement possible quelque soit sa position de départ ? ou comment placer des *wards* pour couvrir la carte de *League of Legends* ?

Pour tout $S \subseteq V(G)$ 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 par un algorithme exhaustif il faut *a priori* un temps :

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

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

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). Pour nos trois problèmes, le certificat est un sous-ensemble de sommets. 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 2.9). La liste peut être finie mais pas connue de manière explicite. C'est le cas de 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 ($k \leqslant 5$) et n pas trop grand ($n \leqslant 50$). Une complexité en $n^{k+1} = n^6$ à 1 GHz prend déjà 15 s pour $n = 50$ et cela passe à 15 minutes pour $n = 100$.

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. Il y a $n!$ ordres et autant de mélanges à vérifier, n étant le nombre de cartes, tout comme le nombre de tournée à vérifier pour l'algorithme naïf du Voyageur de commerce pour n villes.

Supposons maintenant que cet algorithme est implanté sur une architecture massivement parallèle de 10^{41} processeurs surpuissants [3], chacun capable de tester un milliard de mélanges par seconde, et qui est exécuté pendant toute la durée de vie de l'Univers (≈ 13.8 Ma). Combien, dans ces conditions, pourrait-on trier de cartes ?

Réponse : moins de $n = 52$, car le nombre d'ordres possibles que l'on pourra vérifier sur cette durée et à l'aide d'une telle machine serait au plus :

$$(10^{41} \cdot 10^9) \cdot (13.8 \times 10^9) \cdot (365 \cdot 24 \cdot 3600) \geqslant n! \quad \text{seulement si } n < 52.$$

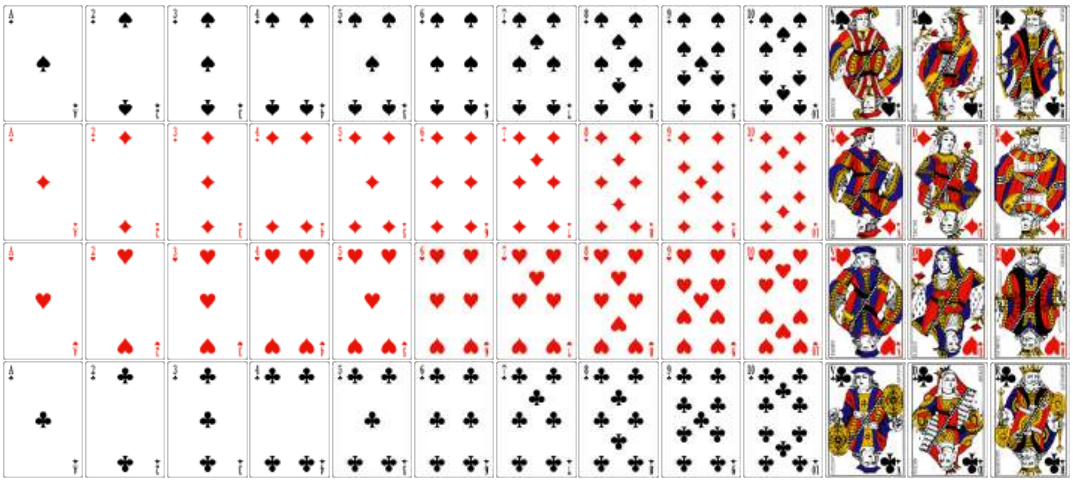
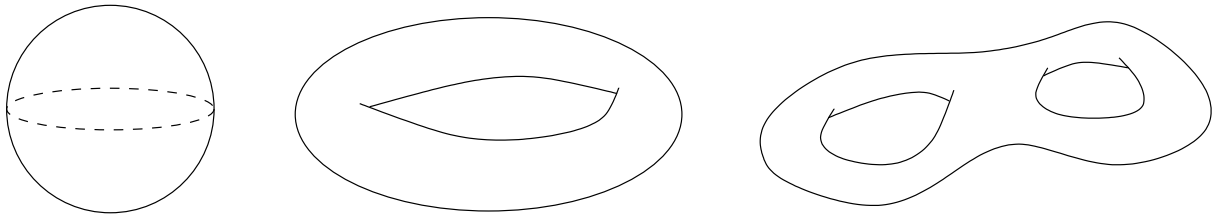


Figure 2.5 – La méthode exhaustive ne permet pas de trier un jeux de 52 cartes, même pour une machine surpuissante. Voir également la vidéo *The Incomprehensible Scale of 52!* consacrée à l'immensité de $52! \approx 8 \times 10^{67}$.

3. Le nombre d'Avogadro, soit le nombre d'atomes dans 12g de carbone ^{12}C, vaut 6.022×10^{23}. Donc si un processeur était un simple atome de carbone, une machine de 10^{41} processeurs pèserait environ $10^{43-23-3} = 10^{20}$kg, le poids de la terre étant estimé à 10^{24}kg. Pour information on estime à 10^{80} le nombre d'atomes de l'Univers et à 2^{400} le nombre total de particules élémentaires, quarks, etc.

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 [4] de la surface sur lequel est dessiné le graphe (0 trou pour les graphes planaires, 1 trou pour les graphes toriques, ... cf. la figure 2.6).



sphère $g = 0$ tore $g = 1$ double tore $g = 2$

FIGURE 2.6 – Surface de genre g obtenue à partir d'un sphère à g trous. Les graphes de genre g peuvent être dessinés sur une surface de genre g sans croisement d'arêtes.

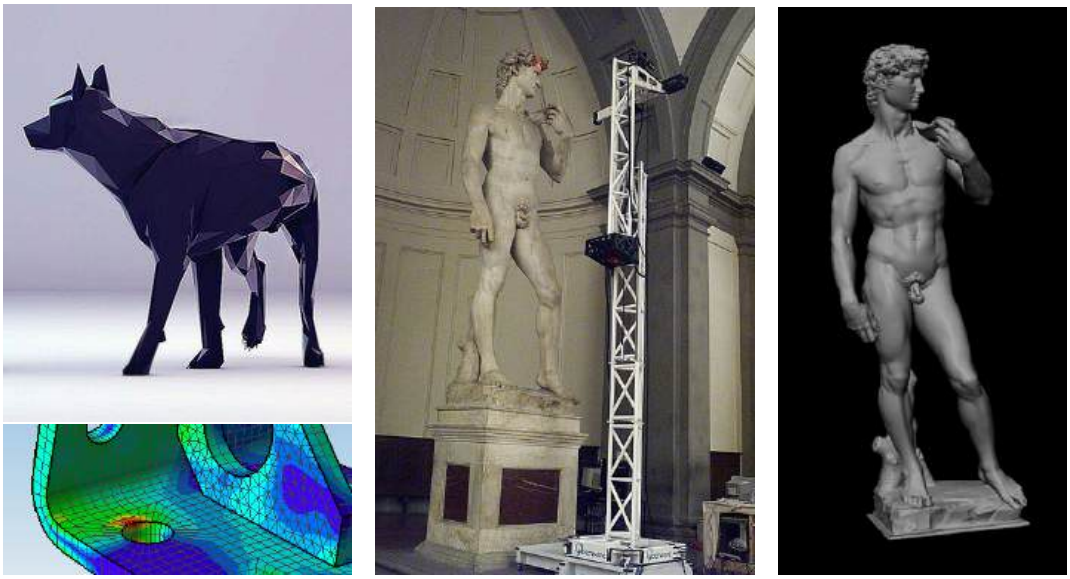
Dans la pratique n est très grand ($n > 10^6$) et on espère que le paramètre soit petit. Typiquement, la triangulation surfacique d'objets réels (une carrosserie de voiture, d'avion ou d'une statue, voir figure 2.7) est un graphe ou maillage comprenant de nombreux sommets ($n \approx$ #polygones [5]) alors que le nombre de trous de la surface supportant le graphe est limité. Très souvent il s'agit d'ouvertures : fenêtre, porte, coffre, phare, toit ouvrant 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 2.8).

Définition 2.1 (Fixed Parametrized Tractable) *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 $k \leqslant n^{O(1)}$. Par exemple, si k représente la taille d'une couverture par sommets, d'un ensemble indépendant ou encore d'un ensemble dominant, on a $k \leqslant n$. Alors, si Π est FPT pour le paramètre k, la fonction f est très certainement exponentielle en k. Car sinon $f(k) \leqslant k^{O(1)}$ impliquerait que $f(k) \cdot n^{O(1)} \leqslant \left(n^{O(1)}\right)^{O(1)} \cdot n^{O(1)} \leqslant n^{O(1)}$ ce qui n'est pas possible sauf si P=NP.

4. De manière équivalente il s'agit aussi du nombre d'anses qu'il faut rajouter à une sphère.
5. Comme les polygones sont généralement des triangles ou des carrés, on confond le nombre de sommets avec le nombre de polygones.



(a) (b) (c)

FIGURE 2.7 – (a) Objets modélisés par des polygones d'une surface de petit genre. (b) Numérisation du « David » de Michel Ange avec 4 milliards de points espacés de 0.25mm, et (c) un rendu avec un maillage de 8 millions de polygones (soit une précision de 4mm) – © *The Digital Michelangelo Project*. Le graphe induit par ces polygones est de genre $\leqslant 3$.

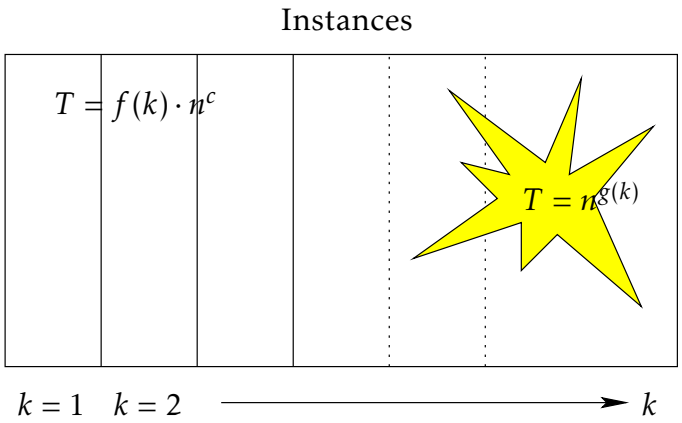


FIGURE 2.8 – Confinement de l'explosion combinatoire grâce à la paramétrisation.

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

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

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

ENSEMBLE DOMINANT

Instance: un graphe G et un entier k
Paramètre: k
Question: est-ce que G possède un ensemble dominant de taille k ?

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] [6]. Cependant, comme on le verra dans la suite du cours, le problème ENSEMBLE DOMINANT est FPT pour le paramètre formé du couple $(k, \Delta(G))$ 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 2.9).

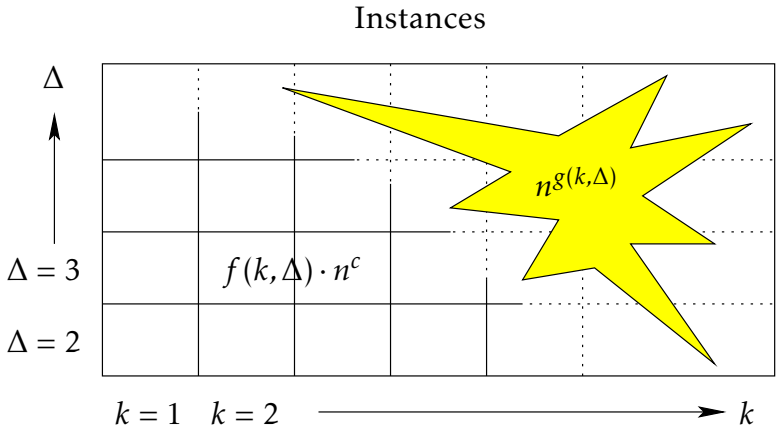


FIGURE 2.9 – Double paramétrage pour ENSEMBLE DOMINANT.

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 ?

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 impliquerai que la 3-COLORATION serait polynomial. Or c'est un problème NP-complet. Cependant, on verra dans la section 2.10 que le même problème, mais

6. En fait il a été démontré qu'on ne peut pas faire mieux que $\Omega(n^k)$ sous la *Strong Exponential Time Hypothesis* (voir la section 2.5 ci-après).

paramétré différemment, est FPT en la *largeur arborescente* du graphe. La largeur arborescente de G est notée $\mathrm{tw}(G)$ pour *tree-width*. Le problème suivant est donc FPT :

k-COLORATION

Instance: un graphe G et un entier k
Paramètre: $\mathrm{tw}(G)$
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 (Nešetřil et Poljak 1985 [NP85]) est $O(n^{\omega k/3}) = O(n^{0.79k})$ où [8] $\omega < 2.3729$. 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 :

$$\text{COUVERTURE PAR SOMMETS} \prec \text{ENSEMBLE INDÉPENDANT} \prec \text{ENSEMBLE DOMINANT}\ .$$

2.5 À propos de SAT

Faisons une petite parenthèse sur le célèbre problème de *satisfiabilité*, SAT, consistant à déterminer s'il existe une affectation de variables booléennes rendant une formule booléenne donnée vraie.

Pour le problème 3-SAT, où chaque clause a au plus trois variables, il existe un algorithme en $O(1.49^n)$ où n est le nombre de variables de la formule. Pour la version générale, SAT, avec des clauses arbitraire et pas seulement limitées à trois variables [7], on ne sait pas faire mieux que l'algorithme naïf en $\Omega(2^n)$ consistant à vérifier toutes les affectations possibles des n variables, même si on peut faire $O(1.24^m)$ et $O(1.08^\ell)$ en fonction du nombre de clauses m et de littéraux ℓ. En fait, ce problème est tellement au cœur de la théorie de la complexité que l'hypothèse selon laquelle aucun algorithme ne peut faire « mieux » que 2^n est connue sous le nom de SETH pour *Strong Exponential Time Hypothesis*. Ici « mieux » signifie une complexité en temps qui serait en c^n avec une constante $c < 2$. Mais la réalité est qu'on est extrêmement loin de pouvoir la prouver. Voici d'ailleurs ce qui est considéré comme le meilleur résultat connu en ce qui concerne le temps minimum (borne inférieure) pour résoudre ce problème :

Théorème 2.1 ([Wil10]) *Tout algorithme déterministe résolvant* SAT *sur une formule à n variables et utilisant un espace mémoire* [8] $n^{o(1)}$, *nécessite un temps d'au moins* $n^{2\cos(\pi/7)-o(1)} \approx n^{1.80193\ldots}$.

7. On peut toujours transformer une formule SAT en formule 3-SAT, mais malheureusement le nombre de variables, n, est un peu plus grand.

8. Le terme « $o(1)$ » représente une fonction de n qui tend vers 0 quand $n \to +\infty$. Donc $n^{o(1)}$ est bien plus petit que $\sqrt{n} = n^{0.5}$ par exemple.

En fait, ce qu'il a été montré est que le temps $t(n)$ et l'espace $s(n)$ de tout algorithme résolvant Sat doit vérifier $t(n)\cdot s(n) \geqslant n^{2\cos(\pi/7)-o(1)}$. Et donc pour un espace $s(n) = O(\sqrt{n})$ par exemple, la borne inférieure sur $t(n)$ n'est seulement de $n^{1.30193\ldots}$.

Une borne inférieure en n^2 serait considérée comme une avancée majeure tout comme ne pas mettre de restriction sur la mémoire (ici sous-linéaire). En fait, il a été prouvé plus tard [WB15] que les techniques de preuves générales développées dans [Wil10] (*Alternation-Trading Proofs*) ne peuvent pas battre l'exposant $2\cos(\pi/7)$. Il faut pour cela de nouvelles idées. Les bornes inférieures issues de la complexité des circuits booléens sont toutes aussi faibles (cf. la section 2.6 ci-après).

Notons aussi que nos trois problèmes jouets s'expriment très facilement par une formule Sat, et qu'un solveur Sat efficace pourrait en venir à bout. C'est une autre façon de produire des algorithmes (et des solutions) si on n'a pas d'autres idées. Cette approche est loin d'être sans espoir en pratique car les problèmes dans NP, qui encodent tous les problèmes difficiles (ceux qui sont NP-complets), sont « faciles au début ». En effet, les solveurs actuelles appliquent des stratégies suffisamment subtiles pour venir à bout de la plupart des problèmes sur des instances d'environ 1 000 bits. Pour un problème où l'instance est un graphe, cela correspond à un graphe à $n = 44$ sommets environ, et même $n = 58$ si les instances sont restreintes aux graphes planaires. [*Question*. Pourquoi ?]

L'idée générale est de simuler un algorithme exhaustif (cf. section 2.3) en codant un certificat positif par des variables booléennes, et en exprimant la validité du certificat par une formule représentant les contraintes. L'action du solveur va alors être de tester tous les certificats possibles (assignation de variables booléennes) jusqu'à en trouver qui vérifient les contraintes.

Par exemple, pour Ensemble indépendant de taille k, on utilisera une variable x_i pour chaque sommet $i = 1,\ldots,n$ indiquant si i est dans l'ensemble indépendant ou pas. On cherche donc une solution où $x_1 + \cdots + x_n \geqslant k$ tout en respectant les contraintes d'un ensemble indépendant, à savoir que pour chaque arête $\{x_i, x_j\}$ du graphe, il faut $x_i = 0$ ou $x_j = 0$, soit la clause $\neg x_i \vee \neg x_j$.

Concrètement, pour savoir si un cycle de longueur 5 possède ou non un ensemble indépendant de taille au moins 3, il faut déterminer si la formule suivante est satisfiable :

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_4 \vee \neg x_5) \wedge (\neg x_5 \vee \neg x_1) \wedge (x_1 + x_2 + x_3 + x_4 + x_5 \geqslant 3)$$

En fait, la dernière clause $x_1 + x_2 + x_3 + x_4 + x_5 \geqslant 3$ n'est pas une formule Sat. Mais on peut transformer la contrainte $\sum_{i=1}^{n} x_i \geqslant k$ en une formule Sat en ajoutant $(k+1)(n+1)$ variables $s_{i,j}$ pour $i \in \{0,\ldots,n\}$ et $j \in \{0,\ldots,k\}$, définit par $s_{i,j} = 1$ ssi $x_1 + \cdots + x_i \geqslant j$. On a alors les contraintes suivantes :

$$\begin{aligned} & s_{i,j} \;\Rightarrow\; s_{i-1,j} \vee (s_{i-1,j-1} \wedge x_i) \qquad \forall i \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,k\} \\ \Leftrightarrow \quad & (\neg s_{i,j} \vee s_{i-1,j} \vee s_{i-1,j-1}) \wedge (\neg s_{i,j} \vee s_{i-1,j} \vee x_i) \end{aligned}$$

avec les initialisations :

$$\begin{array}{rcll} s_{n,k} & = & 1 & \\ s_{i,0} & = & 1 & \forall i \in \{0,\ldots,n\} \\ s_{i,j} & = & 0 & \forall i \in \{0,\ldots,j-1\}, \forall j \in \{1,\ldots,k\} \end{array}$$

Le nombre de clauses supplémentaires, incluant les initialisations, est de $2nk + n + 2 + k \cdot (k+1)/2$.

Finalement, dans le cas de notre cycle à $n = 5$ sommets et pour $k = 3$ cela fait 24 nouvelles variables et 43 nouvelles clauses. On constate aussi que les seules clauses à plus de deux variables sont les $2nk$ clauses dues à la transcription de la contrainte $\sum_i x_i \geqslant k$. Dommage car 2-Sat est polynomial.

2.6 À propos des circuits booléens

Toute fonction booléenne d'arité n, c'est-à-dire à n variables booléennes,

$$f : \{0,1\}^n \to \{0,1\}$$

peut être représentée par un *circuit booléen*. Les *entrées* du circuit sont les variables $x_1,\ldots,x_n$, la *sortie* est le résultat $f(x_1,\ldots,x_n)$, et les *portes* sont des fonctions booléennes d'arité bornée, classiquement d'arité un ou deux comme : $\neg$ (non), $\wedge$ (et), $\vee$ (ou), $\oplus$ (ou-exclusif), $\Rightarrow$ (implique), $\equiv$ (égalité), ...

Comme illustré par la figure 2.10, les circuits booléens peuvent être représentés non seulement par une formule, mais aussi par un graphe acyclique. Dans ce cas, les sources (= nœuds de degré entrant 0) sont les entrées. Le puits (= nœud degré sortant 0) est la sortie de la fonction. Tous les autres nœuds sont les portes du circuit, donc de degré entrant au plus deux, mais de degré sortant arbitraire.

La *taille* du circuit est le nombre de portes binaires. Traditionnellement, on ne compte pas les opérateurs unaires qui sont $\{0,1,\mathrm{id},\neg\}$. En effet, l'identité $\mathrm{id} : x \mapsto x$ correspond à un arc du circuit qui est en quelque sorte déjà compté par les portes. On peut aussi remplacer les fonctions constantes $x \mapsto 0$ et $x \mapsto 1$ en ajoutant deux entrées fixes $x_{n+1} = 0$ et $x_{n+2} = 1$. Enfin, on peut inclure l'opérateur $\neg$ directement dans l'opérateur binaire, et ainsi supprimer toutes les négations. Par exemple, on remplacera $\neg(x \oplus y)$ par $x \equiv y$, et $\neg x$ par $x \oplus 1$. Du coup chaque porte a exactement deux arcs entrants.

La complexité d'une fonction booléenne f est alors la taille minimum d'un circuit l'implémentant. L'ensemble des entrées pour lesquelles la fonction s'évalue à 1 définit un langage. C'est en quelque sorte les entrées acceptées par f. Plus précisément, il s'agit de l'ensemble

$$f^{-1}(1) := \{(x_1,\ldots,x_n) \in \{0,1\}^n : f(x_1,\ldots,x_n) = 1\} .$$

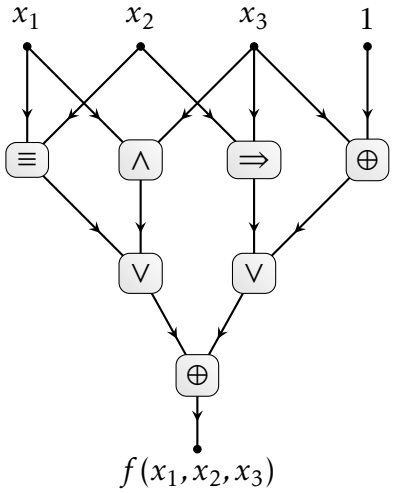


FIGURE 2.10 – Circuit booléen de taille 7 implémentant la fonction booléenne f définie par $f(x_1, x_2, x_3) = ((x_1 \equiv x_2) \vee (x_1 \wedge x_3)) \oplus ((x_2 \Rightarrow x_3) \vee (x_3 \oplus 1))$. Pour une même fonction, plusieurs formules ou circuits sont possibles.

Claude Shannon [Sha49] dans les années 1940 a donné une borne inférieure de $\Omega(2^n/n)$ sur la complexité des fonctions booléennes d'arité n. Elle peut être facilement obtenue en comptant d'un coté le nombre $F(n)$ de fonctions booléennes d'arité n et d'un autre le nombre $C(n,t)$ de circuits booléens à n entrées et de taille t. Bien évidemment, $C(n,t)$ augmente avec t, si bien que si t est « trop petit », alors $C(n,t) < F(n)$. Il y aura alors des fonctions booléennes f qui n'ont pas de circuits booléens de taille t (principe du lemme des pigeons). Cela implique une borne inférieure sur t.

Le calcul de $F(n)$ est relativement simple car une fonction booléenne à n variables peut être représentée de manière unique par un tableau, donnant pour chaque entrée possible la valeur 0 ou 1, soit un mot de 2^n bits puisqu'il y a 2^n entrées possibles (voir l'exemple ci-dessous pour $n = 2$).

$x_1 x_2$	00	01	10	11
$f(x_1, x_2)$	0	1	1	0

Il y a autant de fonctions booléennes que de mots binaires différents de 2^n bits, et donc

$$F(n) \;=\; 2^{2^n}\,.$$

Pour le calcul de $C(n,t)$, on observe que dans un circuit booléen il y a exactement $F(2) = 16$ choix possibles pour chaque porte qui peut de plus prendre ses deux entrées soit parmi les $t-1$ autres portes, soit parmi les n variables, soit parmi les constantes 0 ou 1. Au total cela fait au plus $16 \cdot (t+n+1)^2$ choix pour chaque porte du circuit. Et donc

$$\begin{aligned} C(n,t) &\leqslant \left(16 \cdot (t+n+1)^2\right)^t \;=\; \left(4^2 \cdot (t+n+1)^2\right)^t \;\leqslant\; (4t+4n+4)^{2t} \\ \Leftrightarrow \quad \log_2 C(n,t) &\leqslant 2t \cdot \log_2(4t+4n+4) = 2t \cdot \log_2(4(t+n+1))\,. \end{aligned}$$

Montrons que si $t \leqslant 2^{n-1}/(n+1)$ et $n \geqslant 6$, alors $C(n,t) < F(n)$ comme souhaité. Notons d'abord que si $n \geqslant 6$, alors $n+1 < 2^{n-1}/(n+1)$ (car $(6+1)^2 = 49 < 64 = 2^6$). On en déduit que $t+n+1 < 2\cdot 2^{n-1}/(n+1) = 2/(n+1)\cdot 2^{n-1} < 2^{n-1}$ puisque $n > 1$. Il suit que :

$$\begin{aligned} \log_2 C(n,t) &\leqslant 2t\cdot\log_2(4(t+n+1)) < 2\cdot\frac{2^{n-1}}{n+1}\cdot\log_2\left(2^{n+1}\right) \\ &< \frac{2^n}{n+1}\cdot(n+1) = 2^n = \log_2 F(n) \\ \Leftrightarrow \quad C(n,t) &< F(n)\,. \end{aligned}$$

En dépit de cette borne inférieure exponentielle en n on ne connaît aucune fonction de NP qui soit de complexité super-linéaire, c'est-à-dire une famille $\{f_n\}$ de fonctions booléennes d'arité n telles que $f_n^{-1}(1) \in$ NP pour chaque n. Et il semble ainsi hors de portée d'en prouver une qui soit super-polynomiale, ce qui démontrerait P≠NP. Une partie de la difficulté est que le nombre de fonctions de NP est trop faible pour qu'un argument de comptage simple, comme celui de Shannon, puisse être appliqué.

Il est clair qu'une fonction comme $f(x_1,\ldots,x_n) = (x_1 \equiv x_2)\wedge\cdots\wedge(x_{n-1}\equiv x_n)$, qui détermine si les valeurs x_i sont toutes égales ou pas, est dans NP et nécessite un circuit d'environ n portes [*Exercice.* Pourquoi ?] En fait, on n'a pas beaucoup mieux. La meilleure borne inférieure connue pour la taille d'un circuit booléen de NP est de $(3+1/86)\cdot n - o(n) > 3.01n$ [FGHK16]. Cet un progrès de moins de 0.4% en 30 ans d'efforts après la borne inférieure de $3n-3$ de Norbert Blum en 1984 [Blu84].

La borne en $3n-o(n)$ a en fait été simplifiée et (re)démontrée depuis par plusieurs techniques différentes sans pouvoir la dépassée jusqu'en 2016 avec $3.01n$. Si les opérateurs se limitent à $\{\wedge,\vee,\neg\}$, soit tous les opérateurs binaires saufs $\{\oplus,\equiv\}$, une meilleure borne de $5n-o(n)$ a été démontrée [IM02]. Il existe aussi des bornes inférieures sur la taille des formules (pour des fonctions explicites dans P et donc NP) qui sont respectivement en $n^{2-o(1)}$ [Neč66] et $n^{3-o(1)}$ [Hås98][Tal14] suivant les ensembles d'opérateurs binaires considérés. Cette dernière borne cubique est essentiellement optimale à cause de la formulation 3-Sat de tout problème dans NP.

2.7 Arbre borné de recherche

Idée est de décomposer l'algorithme en deux parties :

1. Construire un espace de recherche qui est souvent un arbre 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 $\leqslant k$ » (resp. « de taille $\geqslant k$ ») sont équivalents tant que k reste polynomial en 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.

2.7.1 Couverture par sommets de taille k

Théorème 2.2 (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 $\mathsf{VC}_1(G,k)$

Entrée: un graphe G et un entier k
Sortie: Vrai si et seulement si G possède une couverture par sommets de taille $\leqslant k$.

1. Si $E(G) = \varnothing$, 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 $\mathsf{VC}_1(G_1, k-1) \vee \mathsf{VC}_1(G_2, k-1)$.

Montrons que l'algorithme est valide. Ce qui n'est pas immédiatement trivial c'est que les choix arbitraires de la ligne 3 mènent 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 de recherche (l'arbre) et 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 $\mathsf{VC}_1(H, k-1)$ est correct pour tout graphe H. Il y a deux cas : G possède ou ne possède pas de couverture par sommets de taille $\leqslant k$. Considérons une arête quelconque de G, disons $\{u,v\}$.

Si G possède une couverture C de taille $\leqslant k$, 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 $\leqslant k-1$ si bien que $\mathsf{VC}_1(G_1, k-1)$ est Vrai. Il en va de même si $v \in C$. Et donc $\mathsf{VC}_1(G_1, k-1)$ ou $\mathsf{VC}_1(G_2, k-1)$ est Vrai.

Si G ne possède pas de couverture par sommets de taille $\leqslant k$, 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

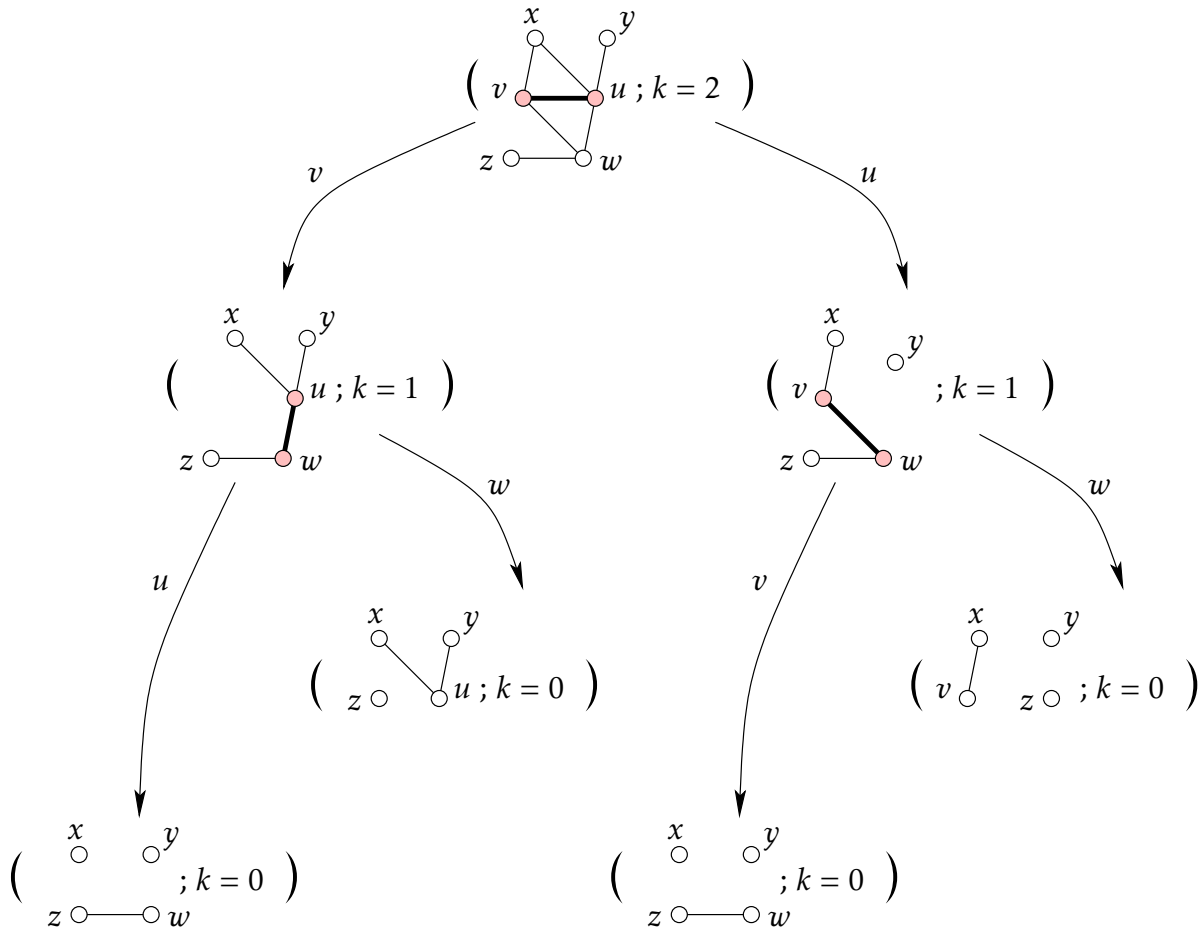


FIGURE 2.11 – Arbre de recherche à 7 nœuds lors d'un appel à l'algorithme $\mathsf{VC}_1(G, 2)$ où G est le graphe à 6 sommets dessiné à la racine. C'est aussi l'arbre des appels de la fonction $\mathsf{VC}_1(G, 2)$.

v constituerait une couverture de taille $\leqslant k$ pour G. Autrement dit, $\mathsf{VC}_1(G_1, k-1)$ et $\mathsf{VC}_1(G_2, k-1)$ sont FAUX tous les deux.

Dans les deux cas la ligne 5 est justifiée.

Calculons la complexité de $\mathsf{VC}_1(G, k)$. L'instruction la plus coûteuse (en dehors des appels récursifs) 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 $\mathsf{VC}_1(G, k)$. 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 $a_0 = 0$ et $a_k \leqslant 1 + 2a_{k-1}$ pour tout $k \geqslant 1$. (En fait $a_k = 1 + 2a_{k-1}$.) Il suit :

$$\begin{aligned} a_k &\leqslant 1 + 2 \cdot (1 + 2a_{k-2}) = 2^0 + 2^1 + 2^2 a_{k-2} \\ &\leqslant 2^0 + \cdots + 2^{i-1} + 2^i a_{k-i} \qquad \text{(pour tout } i \geqslant 1\text{)} \\ &\leqslant 2^0 + \cdots + 2^{k-1} + 2^k a_0 \qquad \text{(pour } i = k\text{)} \\ &\leqslant 2^0 + \cdots + 2^{k-1} = 2^k - 1 \end{aligned}$$

D'où au total une complexité de $O(2^k \cdot (n+m))$. Remarquons qu'on a compté le nombre de nœuds internes (sans les feuilles donc [9]) de l'arbre des appels qui est un arbre binaire complet de hauteur k. □

Pour s'apercevoir que le graphe à $n = 10$ sommets de la figure 2.1 n'a pas de couverture de taille $k < 6$, nous aurions dû, par la méthode exhaustive, effectuer $\binom{n}{k} = 10!/5!^2 = 252$ tests. L'algorithme VC1 nécessite seulement $2^5 - 1 = 31$ tests [10]. En quelque sorte, l'arbre fournit un certificat, négatif ici, pour $k < 6$. Au final, trouver un algorithme efficace, c'est trouver une preuve économe en nombre de tests.

2.7.2 Ensemble indépendant pour les graphes planaires

Rappelons que, dans le cas général, Ensemble indépendant de taille k 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 2.3 *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 2.3, 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. L'analyse de l'algorithme donnée dans la preuve se généralise à tout graphe t-dégénéré (voir la définition ci-après), et la complexité en temps pour résoudre Ensemble indépendant de taille k est de $O((t+1)^k \cdot tn)$.

Définition 2.2 (t-dégénéré) *Un graphe* t-dégénéré *est un graphe où tout sous-graphe induit possède un sommet de degré au plus t.*

9. Les feuilles correspondant à $k = 0$ ne contribuent pas à l'instruction 4, puisque $a_0 = 0$.
10. Ici par « tests » on veut dire tests couteux, les tests au niveau des feuilles de l'arbre de recherche étant triviaux et prenant un temps constant.

Ces graphes peuvent ainsi être « épluchés » en enlevant successivement un sommet de faible degré. On reparlera plus longuement des graphes dégénérés page 144.

La formule d'Euler-Poincaré, $n - m + f = 2 - \chi$ où χ est la caractéristique d'Euler de la surface (ici $\chi = 0$ pour le plan), implique que :

Fait 2.1 *Tout graphe planaire à $n \geqslant 3$ sommets possède au plus $3n - 6$ arêtes.*

En effet, chaque arête du graphe *dual* G^* coupe exactement une arête de G (voir la figure 2.12). Dit autrement $|E(G^*)| = |E(G)| = m$. Toute face étant bordée par au moins trois arêtes, $\deg_{G^*}(u) \geqslant 3$ pour tout sommet u du dual (qui est donc une face de G). Et donc, d'après la formule d'Euler-Poincaré $f = 2 + m - n$, il vient :

$$2m = \sum_{u \in V(G^*)} \deg_{G^*}(u) \geqslant 3|V(G^*)| = 3f = 3(2 + m - n) = 3m + 6 - 3n\,.$$

Ce qui implique directement $3n - 6 \geqslant m$. [*Exercice.* Que devient le majorant sur le nombre d'arêtes d'un graphe planaire à n sommets où toutes ses faces sont bordées par au moins t arêtes ?]

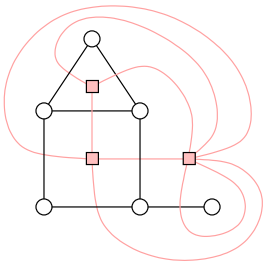


FIGURE 2.12 – Graphe dual G^* (en rosé) d'un graphe planaire G à 6 sommets.

Comme corollaire, on obtient que tout graphe planaire possède un sommet de degré $\leqslant 5$. 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, $\leqslant \frac{1}{n} \sum_u \deg(u) = 2m/n < 6$ d'après le fait 2.1.

Le même argument permet de montrer que les arbres sont 1-dégénérés car $m = n - 1$ dans ce cas. La table page 144 donne une liste de graphes t-dégénérés.

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 $\mathsf{EI}_t(G,k)$

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 $E(G) = \varnothing$ ou si $k = 0$, renvoyer Vrai.
3. Choisir un sommet u_0 de degré $d \leqslant t$ avec pour voisins $u_1, \ldots, u_d$.
4. Pour tout $i \in \{0, \ldots, d\}$, construire le graphe $G_i := G \setminus B(u_i, 1)$.
5. Renvoyer $\bigvee_{i=0}^{d} \mathsf{EI}_t(G_i, k-1)$.

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, $\mathsf{EI}_t(G_0, k-1)$ et $\mathsf{EI}_t(G_1, k-1)$ sont évalués à Faux tous les deux à cause de la ligne 1, $k - 1 = 6$. Et donc $\mathsf{EI}_t(G,k) =$ 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 de recherche dépendant directement du degré des sommets ainsi sélectionnés. Si on ne le dit pas dans la description de l'algorithme EI_t c'est qu'il s'agit d'une optimisation qui n'a pas d'impact sur l'énoncé du théorème 2.3. Compliquer l'algorithme ne peut que compliquer son analyse.

Validité. 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 $\mathsf{EI}_t(H, k-1)$ est correct pour tout graphe H. Soit u_0 un sommet de G ayant pour voisin $u_1, \ldots, u_d$. Il y a deux cas :

Supposons que G possède un ensemble indépendant de taille k. Dans ce cas, il existe 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. Et donc on obtient un tel ensemble J, contenant u_0, 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 $\mathsf{EI}_t(G_i, k-1)$ est Vrai, et donc $\mathsf{EI}_t(G,k)$ 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 possèderait un ensemble indépendant de taille k en prenant celui de taille $k-1$

pour G_i et en ajoutant u_i. Autrement dit $\mathsf{EI}_t(G_i, k-1)$ est Faux pour tous les u_i, et donc $\mathsf{EI}_t(G,k)$ sera évalué à Faux, ce qui est correct.

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

Complexité. L'opération la plus couteuse, 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 $m \leqslant tn$ pour un graphe t-dégénéré, la ligne 3 ne prenant qu'un temps $O(n)$. [*Exercice.* Montrez qu'après un pré-traitement en $O(n)$, on pourrait gérer la ligne 3 en temps $O(1)$.] Notons que pour $t = 0$, le coût est $O(n)$.

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

$$a_k \leqslant (t+1)^0 + (t+1)^1 + \cdots + (t+1)^{k-1} + (t+1)^k \cdot a_0 = \sum_{i=0}^{k-1} (t+1)^i$$

Si $t = 0$, alors le graphe G est 0-dégénéré et donc il n'a pas d'arête. L'algorithme s'arête alors à l'étape 1 ou 2 après un temps $O(1)$.

Si $t > 0$, alors :

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

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

2.7.3 Ensemble dominant pour les graphes planaires

Rappelons que, dans le cas général, Ensemble dominant de taille k 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 2.2). Il en va de même pour un ensemble indépendant S (cf. preuve du théorème 2.3). C'est malheureusement faux pour un ensemble dominant S comme le montre l'exemple suivant (figure 2.13) :

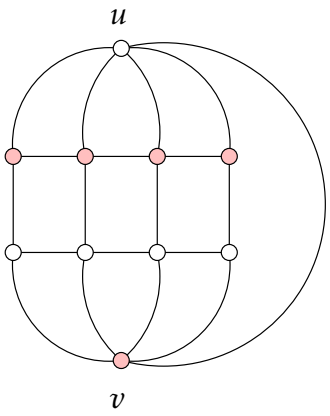


FIGURE 2.13 – Tout ENSEMBLE DOMINANT de taille 2 pour G contient u (et même v). Pourtant ni $G \setminus \{u\}$ ni $G \setminus B(u,1)$ n'ont d'ENSEMBLE DOMINANT de taille 1.

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 2.13), en considérant le sommet u. Si on ne supprime que u, alors le graphe restant n'a pas d'ensemble dominant de taille 1. Si on enlève u et ses voisins, soit $N[u] = B(u,1)$, 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 être 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 cycle $v_1 - v_2 - \cdots - v_{3t} - v_1$ de $3t$ sommets avec $t \geqslant 2$, et dans lequel on ajoute un sommet de degré 1 à v_{3i} pour tout $i \in \{1,\ldots,t\}$ (voir figure 2.14 où $t = 3$). Alors, la solution optimale est de taille t et consiste à choisir les sommets v_{3i} de degré 4. Choisir u de degré $3t \geqslant 6$ ne permet pas de couvrir les feuilles et produit une solution de taille $t+1$ ce qui n'est pas optimal. Le même phénomène se produit avec un $K_{1,t}$ où chaque arête est subdivisée en deux. Notons que dans ces deux exemples, prendre le sommet de degré minimum n'est pas un meilleur choix.

Pour avoir des propriétés plus fortes, on va donc résoudre un problème un petit peu plus général où une partie des sommets du graphe d'entrée sont marqués pour indiquer qu'ils sont déjà dominés. On parle de graphe « annoté ». Plus formellement :

ENSEMBLE DOMINANT ANNOTÉ

Instance: un graphe G avec des sommets marqués et un entier k
Question: est-ce que G possède un ensemble de sommets S de taille k tel que tout sommet non marqué soit dans S ou a un voisin dans S ?

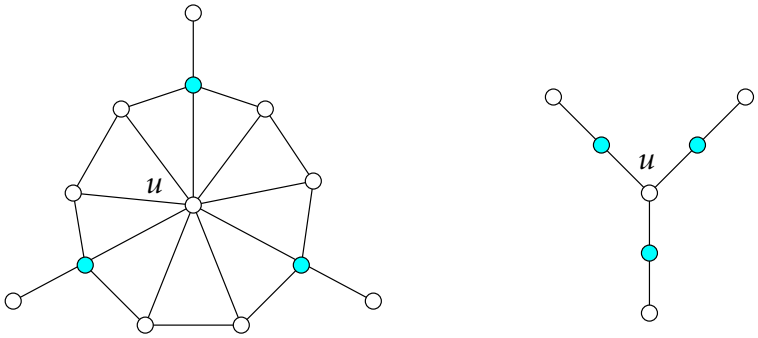


FIGURE 2.14 – Prendre un sommet de degré maximum (ou minimum) n'est pas toujours une stratégie optimale pour ENSEMBLE DOMINANT.

Il s'agit donc de « dominer » les sommets non marqués par des sommets qui peuvent ou pas être marqués. Évidemment, pour résoudre ENSEMBLE DOMINANT, on résout ENSEMBLE DOMINANT ANNOTÉ sur un graphe sans sommet marqué. On verra plus tard (proposition 2.3) que le problème ENSEMBLE DOMINANT ANNOTÉ devient est FPT en k et en $|W|$ où W est l'ensemble des sommets non marqués. Ce problème ainsi paramétré apparaît comme l'étape 2a d'un algorithme plus général, Progress.

Théorème 2.4 ([AFF+05]) *ENSEMBLE DOMINANT ANNOTÉ de taille k pour les graphes planaires à n sommets peut être résolu en temps $O(8^k n^2)$.*

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

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 $u_1, \ldots, u_d$ doit l'être. Sinon on viole la définition d'ensemble dominant. Ainsi, pour chaque $i \in \{0, \ldots, d\}$, en supprimant u_i de G et marquant ses voisins, on obtient un algorithme dont on peut facilement vérifier la validité. [*Exercice.* À faire.]

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. Il faut aussi impérativement faire un branchement qui diminue à coup sûr le paramètre, sous peine de ne pas contrôler la hauteur de l'arbre de recherche. Il faut pouvoir dire que soit u_0 est dans l'ensemble soit l'un de ces voisins l'est. Et ce n'est pas forcément vrai si u_0 est marqué comme le montre la figure 2.15. On est donc condamné à brancher sur des sommets u_0 à dominer. [*Exercice.* Montrer que le branchement sur un sommet à dominer diminue forcément le paramètre k d'une unité.] Si on fait un branchement sans diminuer la taille du paramètre k, alors la taille de l'arbre n'est plus borné par une fonction (même exponentielle) en k mais en n : on va tout droit vers une complexité exponentielle en n.

Avec les divers appels récursifs, il pourrait arriver que les sommets ainsi marqués

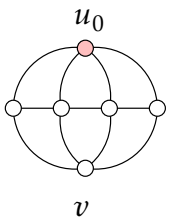


FIGURE 2.15 – On ne peut pas faire de branchement sur le sommet marqué u_0. Aucun sommet de $N[u_0] = \{u_0\} \cup N(u_0)$ ne permet de dominer tous les sommets non marqués, alors que v le peut. Or, $v \in N[w]$ pour tout sommet w à dominer, c'est-à-dire non marqué. Donc en branchant sur un sommet à dominer, on trouvera v.

soient tous les sommets de degré $\leqslant 5$. Si bien que les sommets à dominer restant pourraient être tous de degré > 5. Cela pourrait être le cas du graphe de la figure 2.16, mais aussi de n'importe quel graphe où de manière maligne on attacherait un chemin de longueur 2 à chaque sommet de faible degré. Les branchements sur les sommets de degré 2 de ces chemins (on va voir qu'il est toujours intéressant de choisir le voisin d'un sommet de degré 1), leur suppression, et le marquage du voisin, laisseront ce graphe annoté sans aucun sommet non marqué de faible degré.

En fait la situation peut être assez grave puisque un graphe planaire peut très bien contenir un arbre couvrant contenant k sommets de degré n/k. Le branchement sur chacun de ces k sommets va produire un l'arbre de recherche de degré au moins n/k et de profondeur k, soit au moins $(n/k)^k$ solutions à examiner ! un espace de recherche qui n'est plus vraiment borné en k.

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 marqués.

R2 : Supprimer un sommet marqué v dont tous le voisinage est à dominer et est inclus dans le voisinage au sens large (boule) d'un sommet $u \neq v$ (cf. la figure 2.17).

R3 : Sélectionner le voisin v d'un sommet à dominer de degré 1 (supprimer v du graphe, marquer son voisinage, et décrémenter k).

On ajoute parfois la règle suivante :

R4 : Si u est un sommet isolé à dominer, supprimer u et décrémenter k.

Cependant, elle n'est pas nécessaire car de fait, elle est appliquée par l'algorithme DSAP(G,k) ci-après. [*Question.* À quelle étape ?]

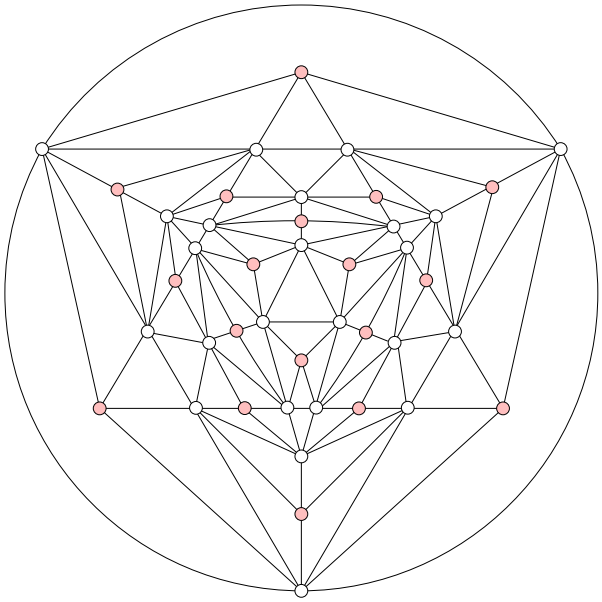


FIGURE 2.16 – Graphe annoté après réduction où tous les sommets à dominer, c'est-à-dire non marqués, sont de degré 7.

Pour implémenter la règle R2 plus efficacement, on peut se limiter aux sommets v de degré 1, 2 ou 3 (voir la figure 2.17).

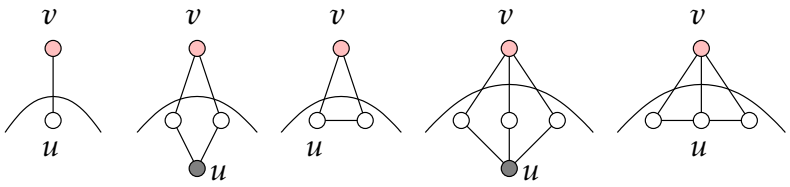


FIGURE 2.17 – Règle R2 limitée aux sommets v de degré $\leqslant 3$. Les sommets grisés peuvent être arbitrairement marqués ou pas.

Dans la suite, on note $\mathrm{R}_i(G,k)$, pour tout $i \in \{1,2,3\}$, la nouvelle instance (G',k') obtenue en appliquant sur l'instance (G,k) la règle de réduction R_i.

Algorithme DSAP(G,k)

Entrée: un graphe G annoté et un entier k
Sortie: Vrai si et seulement si G possède un Ensemble dominant annoté de taille $\leqslant k$.

1. Tant que $k > 0$ et qu'une règle R_i s'applique faire $(G,k) := R_i(G,k)$.
2. Si tous les sommets de G sont marqués, renvoyer Vrai.
3. Si $k = 0$, renvoyer Faux.
4. Choisir un sommet à dominer u_0 de degré $d \leqslant 7$ avec pour voisins $u_1,\ldots,u_d$.
5. Pour tout $i \in \{0,\ldots,d\}$, construire $G_i := G \setminus \{u_i\}$ et marquer les voisins de u_i.
6. Renvoyer $\bigvee_{i=0}^{d} \mathsf{DSAP}(G_i, k-1)$.

Validité. On montre d'abord que les règles de réduction sont correctes (cf. le lemme 2.1 ci-après), ce qui valide la ligne 1. Ensuite, les lignes 3 et 4 sont trivialement correctes en observant qu'après la ligne 1, $k \geqslant 0$. La ligne 4 est justifiée par le lemme 2.2. Enfin, la ligne 6 est justifiée par le fait que dans toute solution S et pour tout sommet à dominer u_0, soit $u_0 \in S$, soit l'un de ces voisins $u_i \in S$. Dit autrement au moins un $u_i \in S$, $i \in \{0,\ldots,d\}$. 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 marqués. 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, dont certains sont peut-être marqués dans G.

Lemme 2.1 *Les règles de réduction sont correctes.*

Preuve. Il est clair que les règles de réduction préservent la planarité. Soit (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 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 marqués ne peut pas modifier l'ensemble des sommets non marqués et 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 marqués v absent de G'. En effet, il suffit de prendre le sommet u contenant le voisinage de v. Le sommet v étant déjà dominé (car marqué), 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 à dominer u de degré 1. Dit autrement G', le graphe G obtenu en enlevant v et en marquant 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 à dominer de G' ainsi que u et v. Donc, c'est une solution de taille k pour G : (G,k) est Vrai. □

Lemme 2.2 *Dans une instance réduite, il existe toujours parmi les sommets à dominer un de degré $\leqslant 7$, même si on limite la règle* R2 *aux sommets marqués de degré $\leqslant 3$.*

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 2.16 montre que la borne sur le degré peut être atteinte car aucune des règles ne peut être appliquée.

Complexité. Le nombre de nœuds de l'arbre des appels est au plus $1 + 8 + 8^2 + \cdots = \sum_{i=0}^{k-1} 8^i = (8^k - 1)/(8 - 1) < 8^k$. La ligne la plus coûteuse est la ligne 1, les autres coutant $O(n)$. Chaque règle réduit l'instance d'au moins un sommet ou une arête. Donc on applique au plus $n + m = O(n)$ règles. En limitant la règle R2 aux sommets de degré 3, comme dans le lemme 2.2, chaque règle se calcule en temps $O(n)$. Cela fait un total de $O(n^2)$ pour la ligne 1. Au total la complexité de l'algorithme est $O(8^k n^2)$.

[*Exercice*. Montrez que la réduction de l'instance peut être effectuée au total en temps $O(n)$ en modifiant éventuellement les règles.] □

Dans les sections 2.10.5 et 2.10.6, nous verrons d'autres algorithmes pour Ensemble dominant pour les graphes planaires, de meilleure complexité mais pas forcément plus efficace en pratique. On verra aussi dans la section 2.12 un autre algorithme, par forcément de meilleure complexité théorique, mais simple à implémenter. Il se révèle très efficace en pratique pour les graphes planaires et même d'autres familles de graphes plus générales comme les carrés de graphes planaires et même les puissances de graphes nul part denses (voir la section 2.12.2). Il permet aussi de résoudre des généralisations du problème Ensemble dominant annoté.

[*Exercice*. Montrez que si F est une forêt, alors l'algorithme DSAP(F,k) a une complexité polynomiale. Quelle est sa complexité ? – Aide : On n'atteint jamais la ligne 4. Proposez un algorithme linéaire pour trouver un ensemble dominant de taille minimum.]

[*Exercice*. Montrez qu'il existe une transformation d'un graphe annoté G en G' tel que Ensemble dominant annoté de taille k pour G est vrai si et seulement si Ensemble dominant de taille $k+1$ est vrai. – Aide : G' possède deux sommets de plus que G.]

2.7.4 Améliorations : raccourcissement de l'arbre

Voici un tableau permettant de mieux comprendre les enjeux de l'explosion combinatoire pour différents algorithmes connus fpt en k pour Couverture par sommets. Le

meilleur algorithme connu pour ce problème explore un espace de recherche de taille bornée par $f(k) = 1.28^k$.

k	$f(k) = 2^k$	$f(k) = 1.47^k$	$f(k) = 1.32^k$	$f(k) = 1.28^k$
10	≈ 1024	≈ 47	≈ 16	≈ 12
20	$\approx 10^6$	≈ 2220	≈ 258	≈ 140
30	$\approx \boxed{10^9}$	$\approx 10^5$	≈ 4140	≈ 1650
⋮	⋮	⋮	⋮	⋮
50	$\approx 10^{15}$	$\approx \boxed{10^8}$	$\approx 10^6$	$\approx 10^4$
75	⋮	$\approx 10^{12}$	$\approx \boxed{10^9}$	$\approx 10^6$
100	⋮	⋮	$\approx 10^{12}$	$\approx \boxed{10^{10}}$
⋮	⋮	⋮	⋮	⋮
500	$\approx 10^{150}$	$\approx 10^{83}$	$\approx 10^{60}$	$\approx 10^{53}$

On considère généralement que le terme $f(k)$ devient un frein conséquent à la complexité d'un algorithme FPT lorsqu'il atteint l'ordre du milliard, soit 10^9 voir 10^{10}. La raison de cette limite somme toute assez arbitraire est que pour un processeur cadencé à 1 GHz, le milliard d'instructions correspond à 1s. Donc une complexité en $f(k) \cdot n^2$ par exemple fait qu'en pratique n sera limité à quelques milliers tout au plus, car un million de secondes de calcul correspondant déjà plus de 10 jours [11].

Théorème 2.5 *Couverture par sommets de taille k pour les graphes à n sommets et m arêtes peut être résolu en temps $O(1.47^k \cdot (n+m))$.*

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 $\Delta(G) \leqslant 2$, alors une couverture par sommets de taille optimale peut être facilement calculée. En effet, si $\Delta(G) \leqslant 2$, 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 2.18). On peut donc se restreindre à un graphe G ayant au moins un sommet u de degré trois ou plus.

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 (le sommet 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 2.19.

11. 11 jours 13 heures 46 minutes et 67 secondes très exactement.

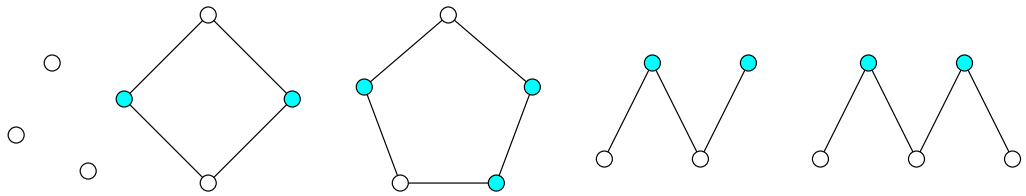


FIGURE 2.18 – Couverture optimale pour un graphe de degré maximum $\Delta \leqslant 2$. La taille optimale est la somme des $\lceil t_i/2 \rceil$ où t_i est le nombre d'arêtes de la i-ème composante connexe. Ici, cette taille vaut $\lceil 4/2 \rceil + \lceil 5/2 \rceil + \lceil 3/2 \rceil + \lceil 4/2 \rceil = 9$.

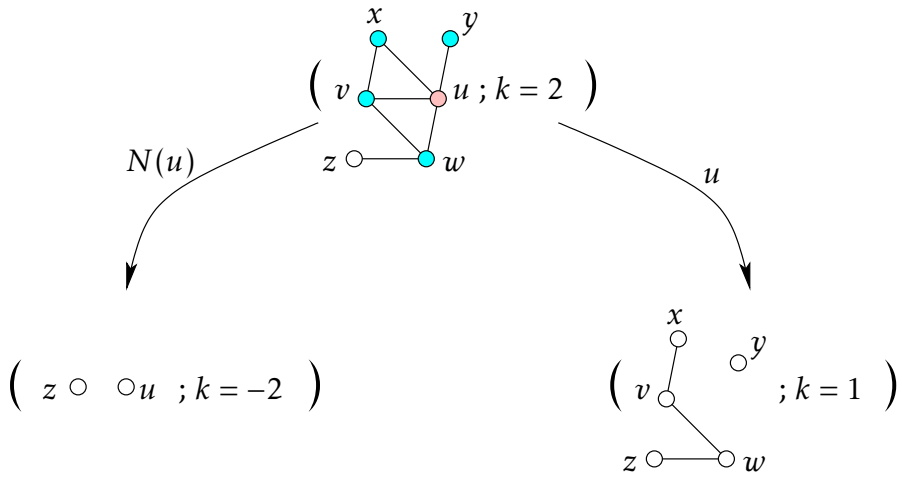


FIGURE 2.19 – Arbre de recherche pour VC2 sur la même instance que la figure 2.11.

Cela mène à l'algorithme suivant, en supposant que la fonction VCdeg2(G,k) qui, étant donné que $\Delta(G) \leqslant 2$, renvoie VRAI si et seulement si G possède une couverture par sommets de taille $\leqslant k$.

Algorithme VC2(G,k)

1. Si $E(G) = \varnothing$ et $k \geqslant 0$, renvoyer VRAI.
2. Si $k \leqslant 0$, renvoyer FAUX.
3. Si $\Delta(G) \leqslant 2$, renvoyer VCdeg2(G,k).
4. Choisir un sommet u de G de degré $\geqslant 3$.
5. Construire $G_1 := G \setminus \{u\}$ et $G_2 := G \setminus N(u)$.
6. Renvoyer VC2$(G_1, k-1) \vee$ VC2$(G_2, k - \deg_G(u))$.

Validité. 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, si

$k < \deg(u)$, alors le second paramètre « k » de la fonction deviendra strictement négatif et on renverra Faux à cause des lignes 1 et 2, ce qui est bien correct. [*Exercice*. Finir la preuve de la correction.]

Calculons la complexité de VC2(G,k). L'instruction la plus coûteuse (en dehors des appels récursifs) est celle de la ligne 5 qui prend un temps $O(n+m)$. [*Exercice*. Montrez que les lignes 1 à 4 ne coûtent que $O(n)$.] Soit a_k le nombre maximum de fois que cette instruction est exécutée lors d'un appel à 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, $a_k \leqslant 1 + a_{k-1} + a_{k-3}$ pour tout $k > 2$. On pose c comme la solution de l'équation $c^0 = c^{-1} + c^{-3}$ et $\alpha = \max_{k\in\{0,1,2\}} \left\{(a_k+1)/c^k\right\}$. Pour information, $\alpha \approx 1.34$ et l'expression exacte pour c est [12] :

$$c \;=\; \frac{1}{3} + \frac{1}{6}\left(116 + 12\sqrt{93}\right)^{1/3} + \frac{2}{3}\left(116 + 12\sqrt{93}\right)^{-1/3} \;\approx\; 1.4655\,. \tag{2.1}$$

Montrons par récurrence que $a_k \leqslant \alpha \cdot c^k - 1$, pour tout $k \geqslant 0$.

Pour $k \in \{0,1,2\}$, on a, à cause du max dans la définition de α :

$$\begin{aligned} \alpha &\geqslant (a_k+1)/c^k \\ \Rightarrow \quad \alpha \cdot c^k - 1 &\geqslant a_k\,. \end{aligned}$$

Donc c'est vrai pour $k \in \{0,1,2\}$. Supposons la récurrence vraie jusqu'à $k-1$. On va la prouver pour k, en se rappelant que par définition de c, $c^{-1} + c^{-3} = c^0 = 1$:

$$\begin{aligned} a_k &\leqslant 1 + a_{k-1} + a_{k-3} = 1 + (\alpha \cdot c^{k-1} - 1) + (\alpha \cdot c^{k-3} - 1) \\ &\leqslant \alpha \cdot c^k \cdot (c^{-1} + c^{-3}) - 1 \;=\; \alpha \cdot c^k - 1 \;=\; O(c^k)\,. \end{aligned}$$

D'où au total une complexité de $O(1.47^k \cdot (n+m))$. □

[*Exercice*. Montrez que la constante additive « 1 » ainsi que les premières valeur a_0, a_1, a_2, dans la formule de récurrence $a_k \leqslant 1 + a_{k-1} + a_{k-3}$, n'ont pas d'impact sur l'ordre de grandeur de a_k, c'est-à-dire que $a_k = O(c^k)$ dès que $a_k \leqslant t + a_{k-1} + a_{k-3}$ pour toute constante $t \geqslant 0$ et valeur a_0, a_1, a_2.]

Vu de loin on a donc réduit l'arbre de recherche à un arbre binaire de profondeur k, comme celui du paragraphe 2.7.1, mais avec un déséquilibre entre ses fils : $k-1$ et $k-3$ pour VC2 contre $k-1$ et $k-1$ pour VC1. La récursion fait que le nombre de nœuds devient beaucoup plus faible pour VC2. [*Exercice*. Calculez a_6 puis construisez un arbre des appels pour VC2 ayant précisément a_6 nœuds. Aide : Commencez par construire les arbres plus petits.] En passant, si le déséquilibre avait été $k-1$ et $k-2$, alors on aurait obtenu un espace de recherche en 1.62^k en cherchant la solution positive de l'équation $c^0 = c^{-1} + c^{-2}$ qui vaut $c = (1+\sqrt{5})/2 = 1.61803...$. C'est une récurrence bien connue sur les nombres de Fibonacci faisant intervenir le nombre d'or $\Phi = (1+\sqrt{5})/2$.

12. Expression calculée grâce à Maple™.

2.8 Réduction à un noyau : « kernelisation »

2.8.1 Principe et propriété

L'idée principale de la technique de kernelisation 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. On dit que I' est un « noyau » (*kernel* en Anglais) pour I.

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.

Plus formellement :

Définition 2.3 (Noyau) *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*.*

Il existe une définition alternative pour ces réductions particulières (on parle parfois de *kernalisation* [13]) qui est similaire sauf qu'on n'impose pas que le problème Π soit décidable. On pourrait, par exemple, vouloir enlever des états « simples » d'une machine de Turing, ce qui peut conduire pour certains problèmes indécidables à une réduction. Cependant, pour la proposition 2.1 ci-après, il est nécessaire que le problème soit décidable.

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 utilisé des règles de réduction dans la construction d'ensembles dominants pour les graphes planaires (cf. la preuve du théorème 2.4). Parmi ces règles, une peut être aussi utilisée pour réduire les instances de Couverture par sommets : si $\deg(u) = 1$, supprimer $N(u)$ et décrémenter k. On y reviendra plus tard avec sa généralisation aux couronnes.

Cette technique, si elle s'applique, mène a une complexité en temps avec un facteur additif exponentiel en k plutôt que multiplicatif comme pour les problèmes fpt :

$$T(n,k) \;=\; n^{O(1)} + h(g(k))$$

où h est la complexité du problème décidable Π.

C'est évidemment très intéressant en pratique. La limite de 10^9 à 10^{10} vue au paragraphe 2.7.4 sur le terme exponentiel vole en éclat. Elle peut être poussée plus loin puisque cette constante n'est plus maintenant multipliée par $n^{O(1)}$ mais simplement ajoutée. Tout dépend évidemment de la taille du noyau $g(k)$ et de la complexité h du problème Π. Mais, comme on va le voir, il y a beaucoup d'exemples où $g(k)$ est linéaire

13. Voir les deux définitions possibles ici : celles de Downey-Fellows de 1999 et de Flum-Grohe de 2006.

en k, comme Ensemble dominant dans les planaires ou Couverture par sommets de taille k.

Si de plus Π est FPT en k, alors h prend une forme plus intéressante encore et la complexité pourra s'exprimer comme :

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

À noter que lors d'une réduction à un noyau, une instance (I,k) de paramètre k peut se transformer en instance (I',k') avec $k' \neq k$, en particulier avec $k' > k$. Cependant, comme la taille de I' est bornée par $g(k)$, il suit que k' doit aussi être borné par une fonction ne dépendant que de $g(k)$, donc de k. Autrement dit, le terme $f(k)\cdot g(k)^{O(1)}$ dans l'expression précédente invoque une certaine fonction f qui n'est pas forcément la même que celle utilisée dans la complexité paramétrique du problème FPT.

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

Proposition 2.1 *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 toute instance I de taille n et de paramètre k en la réduisant, en temps $n^{O(1)}$, à une instance I' équivalente de taille $|I'| \leqslant g(k)$. Puisque Π est décidable, car Π possède un noyau (cf. définition 2.3, on peut résoudre I' en temps au plus $h(|I'|) \leqslant h(g(k))$ où h est la complexité de Π. Donc au final, on a résolu I en temps $n^{O(1)} + h(g(k)) \leqslant h(g(k))\cdot n^{O(1)} = f(k)\cdot n^{O(1)}$ avec $f(k) = h(g(k))$, ce qui montre bien que Π est FPT en k.

$\Leftarrow$ Soit I une instance de taille n de Π que l'on suppose FPT en k. On peut donc résoudre I en temps $f(k)\cdot n^{O(1)}$ pour une certaine fonction f. On cherche à montrer que I à un noyau I', c'est-à-dire à réduire I à une instance I' équivalente et de taille au plus $g(k)$. Il y a deux cas :

- Si $f(k) \geqslant n$, alors le noyau I' est l'instance I est elle-même. Sa taille est $|I'| = |I| = n \leqslant f(k)$, et il suffit donc de prendre $g(k) = f(k)$.
- Si $f(k) < n$, alors on peut résoudre I en temps $f(k)\cdot n^{O(1)} = n^{O(1)}$, c'est-à-dire en temps polynomial. Soient J_k^V et J_k^F des instances triviales de Π telles que J_k^V est Vrai et J_k^F est Faux. (Comme exemple, cela peut-être une instance de graphe avec et sans ensemble dominant de taille k, la NP-complétude garantissant l'exsitence de ces deux instances. [*Question*. Pourquoi ?]) Il est possible qu'une de ces deux instances n'existent pas. Notons que ces instances ne dépendent que de k et de Π, mais ni de I ni de n. Et donc leur taille est aussi indépendante de n. Soit $g(k) = \max\left\{|J_k^V|, |J_k^F|\right\}$. On peut donc construire pour I

le noyau I' défini par

$$I' := \begin{cases} J_k^V & \text{si } I \text{ est acceptée} \\ J_k^F & \text{si } I \text{ est rejetée} \end{cases}$$

Cette réduction de I en I' prend un temps polynomial puisqu'on a vu que I peut être résolue en temps polynomial. Et la taille de l'instance réduite est au plus $g(k)$.

Dans les deux cas nous avons montrer que I se réduit à une instance équivalente I' de taille au plus $g(k)$, donc Π possède un noyau. Notez bien que la réduction se base sur J_k^V et J_k^F qui sont indépendantes de I. Leur calcul ne fait donc pas partie de la réduction. Elles existent et sont fixées quand Π et k le sont.

□

Autrement dit, si l'on sait que le problème est FPT en k, cela ne coûte rien (ou presque) de chercher d'abord un noyau pour ce problème, et bien sûr, le plus petit possible. Malheureusement, la proposition 2.1 ne nous dit pas quelles réductions sont à appliquer pour trouver le noyau ...

Remarque. Si un problème paramétré par k peut être résolu en temps $T(n,k) = f(k) \cdot n^{O(1)}$, alors il peut être résolu en temps $g(k) + n^{O(1)}$ pour une certaine fonction $g(k)$. En effet, $2xy \leqslant x^2 + y^2$ puisque $0 \leqslant (x-y)^2 = x^2 - 2xy + y^2$. En particulier, $f(k) \cdot n^c \leqslant \frac{1}{2}(f(k)^2 + (n^c)^2) = g(k) + n^{O(1)}$. Mais cela n'implique pas que le problème a un noyau ! Sans la proposition 2.1, il n'est pas trivial d'affirmer que tout problème de complexité $g(k) + n^{O(1)}$ possède un noyau.

2.8.2 Exemple de noyau

Au lieu de couvrir des lignes (=arêtes) par des points (=sommets), comme dans COUVERTURE PAR SOMMETS, on va essayer de couvrir des points par des lignes ...

COUVERTURE PAR DROITES

Instance: un ensemble P de points du plan et un entier k
Paramètre: k
Question: est-ce que P peut être couvert par k droites ?

Ce problème, *Line Cover* en Anglais, 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 (cf. figure 2.20) : s'il existe une droite couvrant un ensemble S de plus de k points, alors on les supprime de P. L'instance (P,k) se réduit alors en $(P \setminus S, k-1)$.

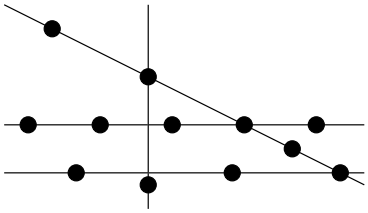


FIGURE 2.20 – Couverture de points du plan par des droites de taille optimale. [*Exercice*. Pourquoi est-elle optimale ?]

Nous avons trois points à démontrer :

La règle de réduction est valide. 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. En effet, la solution de (P,k) contient nécessairement une droite couvrant tous les points de S, puisque sinon $|S| > k$ droites différentes seraient requises (ce qui est exclu). Donc, dans la solution de (P,k), $k-1$ droites couvrent $P \setminus S$, c'est-à-dire $(P \setminus S, k-1)$ est vraie.

La réduction est polynomiale. En effet, on peut supposer sans perte de généralité que chaque droite de la solution couvre au moins deux points, si bien qu'il y au plus $\binom{n}{2} = n(n-1)/2 = O(n^2)$ droites candidates à tester et compter les points de P que contient une droite prend un temps $O(n)$. La réduction est donc polynomiale.

Le noyau est de taille au plus $g(k)$. 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 puisqu'avec k droites d'au plus k points on ne peut couvrir qu'un maximum de k^2 points. On a donc réduit le problème à une instance de taille $g(k) \leqslant k^2$.

On a donc montré que ce problème a un noyau de taille au plus k^2. [*Exercice*. Montrez que le problème de couvrir n points de $\mathbb{R}^3$ par k plans (COUVERTURE PAR DES PLANS) possède un noyau de paramètres (k,t) en supposant qu'il n'y a au plus t points alignés. Quelle est sa taille ? La réduction est-elle polynomiale ? Donner la complexité d'un algorithme brute force permettant de résoudre COUVERTURE PAR DES PLANS de taille k. En utilisant le noyau, en déduire un algorithme efficace pour résoudre COUVERTURE PAR DES PLANS pour (k,t) fixés.]

Déterminer un noyau de petite taille est parfois plus technique. 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 2.7.3. 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, en utilisant des règles calculées et optimisées par ordinateurs, un noyau de taille $43k$ a même été avancé sans être prouvé [Hal16].

Il est intéressant de constater pour ce dernier résultat que c'est un algorithme amont qui détermine et optimise les règles d'un second algorithme. Il est concevable d'imaginer que les algorithmes ainsi optimisés pourraient se révéler efficaces. Par exemple,

pour certains problèmes et au moins certaines instances, l'algorithme pourrait produire un noyau aussi petit que [14] $\log^2 k$, mais que l'analyse de la taille du noyau ainsi produit soit hors de portée. Autrement dit, des problèmes difficiles pourraient posséder des algorithmes efficaces que l'on pourrait calculer sans être capable d'en prouver leur efficacité.

2.8.3 Couverture par sommets

On va utiliser le fait mathématique suivant qui permet de supprimer les termes polynomiaux en k dans une complexité en c^k, ce qui permettra de pouvoir dire par exemple que $1.6^k \cdot k^2 = O(1.7^k)$. Le but de cette notation n'est pas de tricher, mais comme toujours, de raccourcir les énoncés. Cependant, il arrive que la notation « O » est à la complexité ce qu'est un tapis à la poussière.

Fait 2.2 *Pour tout réels $c, \varepsilon > 0$ et tout entiers $k \geqslant i > 0$,*

$$c^k \cdot k^i \ < \left(\frac{ic}{\varepsilon}\right)^i \cdot (c+\varepsilon)^k \ .$$

En particulier, $c^k \cdot k^2 < (2c/0.1)^2 \cdot (c+0.1)^k = 400c^2 \cdot (c+0.1)^k = O((c+0.1)^k)$ pour $c > 0$ constant et tout $k > 0$. Notez bien que $O(2^n \cdot n^2) \neq O(2^n)$. Certains auteurs introduisent parfois la notation $O^*(2^n)$ pour indiquer une complexité en $2^n \cdot n^{O(1)}$ et $\tilde{O}(f(n))$ à la place de $f(n)\text{polylog}(f(n))$ pour supprimer les termes poly-logarithmiques.

Preuve. Par la formule du binôme de Newton, on a $(c+\varepsilon)^k = \sum_{j=0}^{k} \binom{k}{j} c^{k-j}\varepsilon^j$. En utilisant le fait que $\binom{k}{i} \geqslant (k/i)^i$, il vient (l'inégalité stricte ci-dessous vient du fait que $k, i > 0$ et que $\binom{k}{0} c^k \varepsilon^0 > 0$) :

$$\begin{aligned} (c+\varepsilon)^k \ &> \ \binom{k}{i} c^{k-i}\varepsilon^i \ \geqslant \ \left(\frac{k}{i}\right)^i c^{-i}\varepsilon^i c^k \ \geqslant \ \left(\frac{\varepsilon}{ic}\right)^i k^i c^k \\ \Rightarrow \quad \left(\frac{ic}{\varepsilon}\right)^i \cdot (c+\varepsilon)^k \ &\geqslant \ k^i c^k \ . \end{aligned}$$

□

Théorème 2.6 *Couverture par sommets de taille k pour les graphes à n sommets peut être résolu en temps $O(1.47^k + kn)$.*

Preuve. On va montrer que le problème Couverture par sommets de taille k possède un noyau de taille [15] $O(k^2)$, calculable en temps $O(kn)$.

14. Notons qu'une complexité FPT avec $f(k) = 2^{(\log^2 k)} = k^{\log k}$ est ni polynomiale ni exponentielle k, mais entre les deux.

15. On veut dire ici que le nombre de sommets plus le nombre d'arêtes du noyau est $O(k^2)$.

Observation. C'est la même idée que pour Couverture par droites, à savoir que 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. On considère l'algorithme suivant, basé sur l'observation précédente :

Algorithme VC3(G,k)

1. Si $E(G) = \varnothing$ et $k \geqslant 0$, renvoyer Vrai.
2. Si $|E(G)| > k \cdot \Delta(G)$, renvoyer Faux.
3. On pose $H := G$ et $k' := k$.
   Tant que $k' > 0$ et $\exists u \in V(H)$ tel que $\deg(u) > k'$ faire :
   - $H := H \setminus \{u\}$ et $k' := k' - 1$.
4. Si $|E(H)| > k' \cdot \Delta(H)$, renvoyer Faux.
5. Supprimer les sommets isolés de H.
6. Renvoyer VC2(H,k').

Validité. Le test de l'étape 1 est trivialement valide. Le test de l'étape 2 est aussi valide, car un sommet u appartenant à une couverture ne peut couvrir qu'au plus $\deg(u) \leqslant \Delta(G)$ arêtes.

L'étape 3 applique la règle de réduction consistant à sélectionner et supprimer un sommet de degré $> k$. Après l'étape 3, G possède une couverture de taille k si et seulement si H possède une couverture de taille $k' \geqslant 0$.

Le test de l'étape 4 se justifie de la même manière que celui de l'étape 2. Si $|E(H)| > k' \cdot \Delta(H)$, alors une couverture de taille $k' \geqslant 0$ pour H n'existe pas. Et on a bien $k' \geqslant 0$ à cause de la condition du « tant que » de l'étape 3.

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

Complexité. On note m le nombre d'arêtes de G. Il est possible d'avoir $m = \Theta(n^2)$. L'étape 1 prend un temps $O(n)$ en parcourant la liste d'adjacence [16].

L'étape 2 s'implémente en $O(kn)$ de la manière suivante. On compte les arêtes de G tout en veillant à ne pas dépasser $k \cdot (n-1)$. Si $m > k \cdot (n-1)$ il faut bien sûr renvoyer Faux sans même attendre d'avoir calculé $\Delta(G)$ car $\Delta(G) \leqslant n-1$. Dans l'autre cas, $m \leqslant k \cdot (n-1)$. On peut alors calculer $\Delta(G)$ en temps $O(n+m) = O(kn)$ et faire le test plus fin de l'étape 2.

16. On suppose les graphes représentés par liste d'adjacence.

De la même façon, l'étape 4 prend un temps $O(kn)$ et l'étape 5, $O(kn)$. Donc les étapes 1, 2, 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 2, on a $m \leqslant k \cdot (n-1)$. On peut, par exemple, utiliser une structure de données permettant de maintenir les degrés des sommets. On peut le faire sous la forme d'une table, disons `DEG`, indexée par les sommets et construite juste avant l'étape 3. On peut aussi se permettre dans `DEG` de coder par 0 le fait que le sommet soit supprimé ou isolé, ceci en prévision de l'étape 5. La table `DEG` peut être initialisée en temps $O(n+m) = O(kn)$ en un simple parcours [17] de G. La mise à jour de `DEG` se fait en temps $O(\deg_G(u)) = O(n)$ (il faut faire attention de ne supprimer les sommets qu'une fois !) Trouver un sommet de degré $\geqslant k'$ se fait aussi en temps $O(n)$ grâce à `DEG`. Donc, au total l'étape 3 prend un temps $O((k-k') \cdot n) = O(kn)$.

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 dans H. Cela se fait en parcourant G en temps $O(kn)$, et en ne gardant, grâce à `DEG`, que les sommets de degré > 0, ce qui à la volée implémente l'étape 5.

Après l'étape 4, H possède au plus $k' \cdot \Delta(H) \leqslant k'^2$ arêtes à cause de la double condition du « tant que » de l'étape 3, soit $k' = 0$, soit $\Delta(H) \leqslant k'$. Donc après l'étape 5, il possède au plus $2k'^2$ sommets, tout sommet ayant au moins un voisin et une arête étant incidente à au plus deux sommets distincts. En passant, cela montre que le problème possède un noyau avec au plus $2k^2$ sommets puisque $k' \leqslant k$.

Le temps pour l'étape 6 est de $O(c^{k'} \cdot k'^2) = O(c^k \cdot k^2)$ d'après le résultat précédent (théorème 2.5) avec $c \approx 1.4655$ est la constante donnée par l'équation (2.1).

La complexité totale de l'algorithme est donc : $O(kn + 1.466^k \cdot k^2) = O(1.47^k + kn)$ d'après le fait 2.2.

Remarque. On aurait pu montrer que le noyau possède au plus k'^2 sommets (au lieu du double) en supprimant les sommets de degré 1, grâce à une règle comme celle-ci à mettre entre les lignes 4 et 5 de VC3 :

4½. Tant que $k' > 0$ et $\exists u \in V(H)$ avec $\deg(u) = 1$, faire $H := H \setminus N(u)$ et $k' := k' - 1$

Après la ligne 5, on a alors que $|V(H)| \leqslant |E(H)|$ puisque tous les sommets de H sont de degré $\geqslant 2$. Mais avant de faire ce travail d'optimisation, il faut s'assurer qu'elle ne sera pas réalisée comme effet de bord par l'algorithme suivant, ici VC2 à la ligne 6. □

Le meilleur algorithme connu a une complexité [18] de $O(1.2738^k + kn)$ [CKX06]. Il utilise, en plus de la technique d'arbre de recherche bornée, la technique de réduction

17. Les graphes sont toujours, sauf mentions contraires, représentés par listes d'adjacence.
18. La complexité précise est $O(c^k \cdot k + kn)$ où $c \approx 1.2737...$. Et bien sûr par le fait 2.2, $c^k \cdot k = O(1.2738^k)$.

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 2.21). 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, c'est-à-dire sans sommet en commun (voir le paragraphe 3.2.1 pour plus de détails). Une remarque très utile est que toute couverture par sommets doit utiliser au moins un sommet différent par arête du couplage.

Définition 2.4 (Couronne) *Une* couronne *est un ensemble I 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 la taille du couplage, soit $|N(I)|$.*

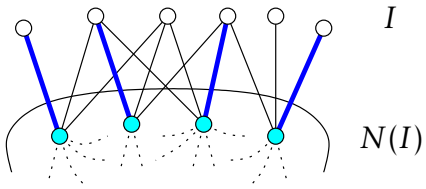


FIGURE 2.21 – Une couronne de taille 4.

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 $|N(I)|$. [*Exercice.* Trouvez la plus grande courrone du graphe de la figure 2.19.]

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

Lemme 2.3 *Tout graphe sans sommet isolé possédant $> 3k$ sommets et une couverture par sommets de taille $\leqslant k$ contient une couronne de taille non nulle qui peut être calculée en temps polynomial.*

Par ce biais on obtient un noyau d'au plus $3k$ sommets en supprimant successivement les couronnes de G. On retrouvera la même construction dans le lemme 3.1 au paragraphe 3.3.3 du chapitre 3 sur les approximations.

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 $|X| = 2|M| \leqslant 2k$ car $|M| \leqslant k$ puisque G possède une couverture par sommets de taille k. Il suit que $|I| > k$, sinon G aurait $\leqslant 3k$ sommets.

On construit le graphe G' induit par les arêtes entre X et I. Il s'agit du graphe G dans lequel les arêtes entres les sommets de X ont été supprimées. Ce graphe est biparti. Comme on le verra plus tard dans le cours (cf. le théorème 3.1 de König-Egavary), 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.

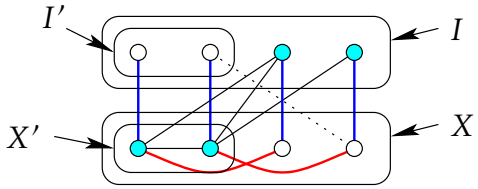


FIGURE 2.22 – Couplage maximal de G (en rouge), couplage maximum de G' (en bleu), couverture minimum (en cyan), et couronne I' avec $N(I') = X'$. L'arête en pointillée ne peut exister.

La couverture C contient au moins un sommet de X. Sinon, cela force $C = I$ qui aurait une taille $> k$, car G (donc I) ne possède pas de sommet isolé. Soit $X' = X \cap C$ (qui possède donc au moins un sommet), et soit I' l'ensemble des sommets induits par les arêtes de M' dont une extrémité est dans X' (cf. figure 2.22). Comme X' n'est pas vide, I' n'est pas vide non plus. 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 de taille non nulle dans G. □

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 flots (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 $O(k^2\sqrt{k^2}) = O(k^3)$ avant d'appliquer la recherche exponentielle en c^k. Cela donne une complexité totale en $O(kn + k^3 + 1.47^k) = O(kn + 1.47^k)$ (car $k^3 = O(1.47^k)$). Notons que trouver une couronne de la plus grande taille possible est polynomial [AKFLS07].

2.8.4 Noyau et approximation

De nombreuse 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 $c \geqslant 1$, et qui soit formé d'un sous-graphe du graphe initial (c'est le cas des réductions qu'on a vues jusqu'à présent). 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 $\leqslant k$? Si oui en renvoyer une de taille $\leqslant ck$, 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 du noyau et de tous ceux sélectionnés lors de la réduction. Si lors de la réduction t sommets ont été sélectionnés, alors le noyau pour une couverture de taille au plus $k-t$ contiendra au plus $c \cdot (k-t)$ sommets. Au total, cela donne une couverture de taille au plus $t + c \cdot (k-t) \leqslant ck$ puisque $c \geqslant 1$.

On en déduit aussi un algorithme polynomial qui fournit une couverture au plus c fois plus grande que la taille optimale, soit une c-approximation (cf. définition 3.1 du chapitre 3), en cherchant (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 2.3 donnant un noyau de taille $2k$ (voir [Khu02]). Comme on le verra par 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 $(2-\varepsilon) \cdot k$ pour Couverture par sommets, à moins bien sûr que P=NP. Mais la borne de $(2-\varepsilon) \cdot k$ 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 2.2 *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. Soit C l'ensemble des sommets de la couleur la plus fréquente. Évidemment, $|C| \geqslant n/4$ et C est un ensemble indépendant. Si $k \leqslant n/4$, 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. □

[*Exercice*. Généraliser aux graphes qui peuvent être coloriés en t couleurs en temps polynomial en montrant qu'Ensemble indépendant de taille minimum pour de tels graphes possède une t-approximation.]

[*Exercice**. À REVOIR ... On a vu dans la remarque de la section 2.8.2, qu'Ensemble dominant de taille k dans les graphes planaires admet un noyau de taille linéaire, disons d'au plus ck sommets. Construire un algorithme d'approximation de ratio $c/2$ pour ce même problème en supposant l'existance d'un noyau qui est un sous-graphe du graphe initial.]

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 $(4/3-\varepsilon)\cdot k$ sommets, pour tout $\varepsilon > 0$, sauf si P=NP.

2.8.5 À propos des noyaux

Il y a une « course » aux noyaux les plus petits possibles pour les problèmes fpt, incluant des compétitions, et des sites où l'on peut trouver les meilleurs résultats connus [19]. Mais au-delà de ces compétitions, il existe des résultats généraux.

Par exemple, 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 on le restreint aux graphes de genre borné [BFL+09]. Ceci doit être rapproché du fait qu'Ensemble dominant n'est *a priori* pas fpt en k.

Il a été aussi montré dans [PRS12] qu'Ensemble dominant de taille k est fpt en (k,i,j) pour les graphes sans $K_{i,j}$ comme sous-graphe. Voir la figure 2.26 pour une représentation des graphes $K_{2,3}$ et $K_{3,3}$. Notons que la famille des graphes sans $K_{3,3}$ contient déjà tous les graphes planaires qui sont sans mineur $K_{3,3}$ donc sans sous-graphe $K_{3,3}$. Le problème ainsi paramétré admet un noyau de taille polynomial en k pour i,j fixés. Le nouveau graphe possède $n' = O((j+1)^{i+1}k^{i^2})$ sommets et le nouveau paramètre est $k' = O(n')$. Les graphes t-dégénérés que l'on a déjà rencontrés sont par exemple sans $K_{t+1,t+1}$, ce qui inclut aussi les graphes de degré maximum t. Et les graphes de degré maximum t n'ont pas de noyau de taille $k^{o(t^2)}$, sauf si P=NP. Avec les mêmes techniques, on peut résoudre Ensemble dominant Indépendant pour les graphes sans $K_{i,j}$, le noyau ayant alors $O(jk^i)$ sommets.

Mais il existe des problèmes plus « coriaces », qui sont fpt et pourtant sans « petit » noyau. C'est le cas de Chemin de taille k, c'est-à-dire déterminer si un graphe possède ou non un chemin simple de k sommets. On parle de longueur et non de taille s'il s'agit

19. http://fpt.wikidot.com/fpt-races

du nombre d'arêtes. Il est FPT en k et peut être résolu en temps $2^k n^{O(1)}$ [Wil09] (voir aussi le paragraphe 116). Cependant, on peut montrer qu'il ne possède pas de noyau de taille polynomiale en k sauf si co-NP $\subseteq$ NP/poly. La classe NP/poly est la classe des problèmes qui peuvent être résolu par un algorithme polynomial non-déterministe (NP) ayant accès à un oracle de taille polynomial dépendant seulement de la taille n de l'entrée. Cet oracle pourrait être, par exemple, un sous-ensemble très particulier d'entiers $\mathcal{O}_n \subset \left\{1,\ldots,n^2\right\}$ donnant la liste des indices des premières machines de Turing qui s'arrêtent. On reviendra sur la classe co-NP au chapitre 3.

2.9 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 milieux 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 polynomial lorsque le paramètre est fixé.

2.9.1 Définition

On rappelle que la contraction d'une arête entre u et v consiste à (voir figure 2.23) :

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 à 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).

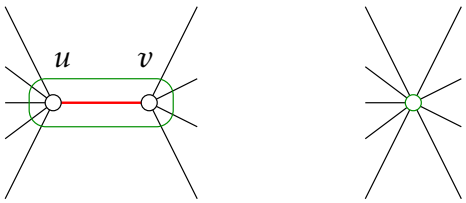


FIGURE 2.23 – Contraction de l'arête $\{u, v\}$.

Définition 2.5 (Mineur) *Un* mineur *d'un graphe G est un sous-graphe d'un graphe obtenu à partir de G en contractant des arêtes.*

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 un nombre arbitraire de fois les trois opérations suivantes sans ordre particulier :

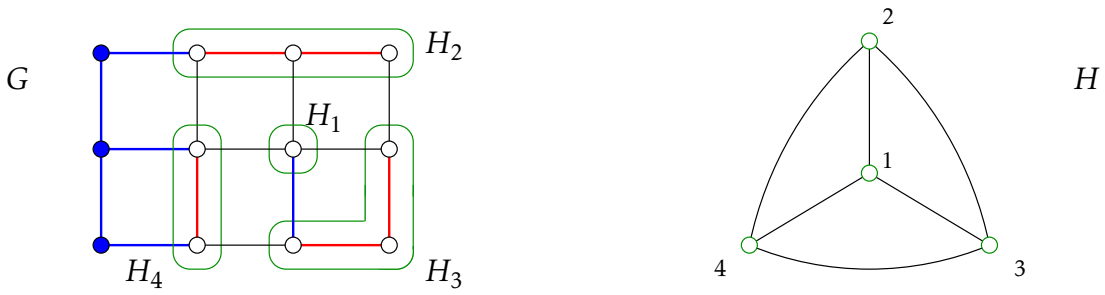


FIGURE 2.24 – Un mineur $H = K_4$ d'une grille G. Les contractions sont en rouges, les suppressions en bleues. $(H_1, \ldots, H_4)$ est un modèle de H dans G. Plusieurs models sont possibles.

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

Une autre façon de montrer que H est un mineur de G, en supposant que $V(H) = \{1, \ldots, h\}$, est de construire ses sous-graphes connexes et disjoints $H_1, \ldots, H_h$ 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 $(H_1, \ldots, H_h)$ est parfois appelée *model* de H dans G (voir la figure 2.24).

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 2.25), tout comme les familles de graphes qui ne sont pas close par sous-graphes, comme les graphes de diamètre au plus k par exemple.

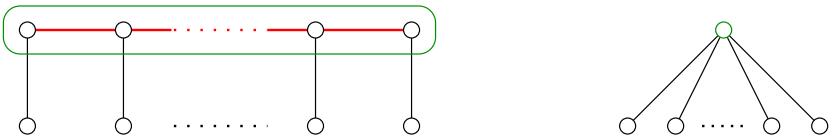


FIGURE 2.25 – La famille des graphes de degré borné n'est pas close par mineurs.

[*Exercice*. Montrez que la famille $\mathcal{D}_3^1$ des graphes ayant, par composante connexe, au plus 1 sommet de degré au moins 3 est close par mineur.]

2.9.2 Théorème des mineurs de graphes

Théorème 2.7 (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 difficulté [20] dans la preuve du théorème 2.7 est 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 2.7 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 $\mathcal{O} = \{K_4\}$. Voir la figure 2.26. Si l'on croisent régulièrement ces familles, c'est peut-être en partie lié à la simplicité de leur caractérisation.

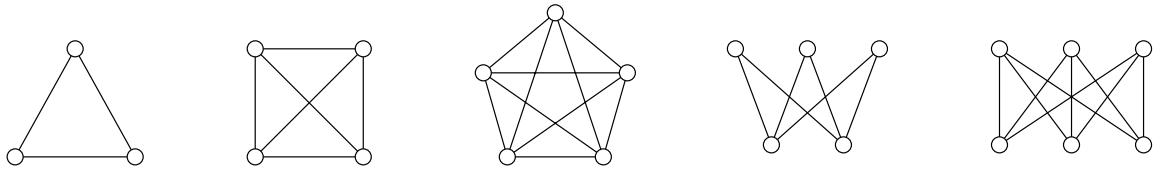


FIGURE 2.26 – Obstructions intervenant dans les familles closes par mineur. De gauche à droite : K_3, K_4, K_5, $K_{2,3}$ et $K_{3,3}$.

[*Exercice*. Trouvez les obstructions pour la famille $\mathcal{D}_3^1$ introduite dans l'exercice page 75.]

Théorème 2.8 ([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 $c(H) \cdot O(n^3)$.*

L'algorithme cubique, pour H fixé, n'est pas du tout efficace en pratique car la constante $c(H)$, résultante de la preuve, est gigantesque. On parle parfois de *Galatic Algorithm* tellement le temps est ridiculement grand en pratique (bien que de complexité théorique modeste) pour des graphes issus de données « terrestres ». En effet, bien que non explicité dans l'article original, il a été estimé [Joh87, p. 289] que la constante vérifiait *a priori* :

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

est une tour d'exponentielles en k de hauteur h. Plus formellement : $\mathrm{tour}(k,0) = k^0 = 1$ et $\mathrm{tour}(k,h) = k^{\mathrm{tour}(k,h-1)}$. Pour $|V(H)| = 2$ cela fait déjà :

$$c(H) \;>\; \mathrm{tour}(2, 3 + \mathrm{tour}(2, \mathrm{tour}(2,1))) = \mathrm{tour}(2, 3 + \mathrm{tour}(2,2)) = \mathrm{tour}(2,7) = 2^{2^{2^{65536}}}.$$

Cela n'a aucun sens puisque le nombre de milliardièmes de secondes (10^{-9}s) écoulés depuis le début de l'Univers (13.8 Ma) est $1.1 \times 26! \approx 4.4 \times 10^{26} \approx 2^{89}$. Et donc même

20. La preuve qui s'étale sur plus de 20 ans (de 1983 à 2004) contient plus de 700 pages réunies dans 20 articles, de *Graph Minors I* à *Graph Minors XX*.

avec autant de processeurs 1 GHz que de particules dans l'Univers (2^{400}) s'éxécutant depuis le début des temps, on ne peut exécuter que $< 2^{489}$ instructions. On est loin du compte. Il est donc plus utile de développer des algorithmes spécifiques que d'appliquer le théorème 2.8.

Pour les familles citées ci-dessus (graphes planaires, planaires extérieurs, 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 [RL08].

Bien sûr la constante $c(H)$ est liée à la preuve du théorème 2.8, 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, sauf si P=NP. Des résultats récents mais non publiés à ce jour montreraient que $c(H)$ est majorée par une double exponentielle en la taille de H. Aussi des algorithmes réputés plus simple en $O(n^2)$ ont été proposés dans [KKR12][GKR13], mais leur complexité cache des constantes dépendantes de H non explicitées.

2.9.3 Applications

Un graphe est *sans entrelacement* (*linkless* en Anglais) s'il est possible de le dessiner dans $\mathbb{R}^3$ sans croisement d'arêtes [21] tel qu'il n'existe pas deux cycles disjoints entrelacés. Deux courbes de $\mathbb{R}^3$ sont entrelacées si leur projection sur $\mathbb{R}^2$ s'intersectent. C'est donc une sorte de « planarité spaciale », car dans le plan, deux cycles disjoints ne se coupent jamais.

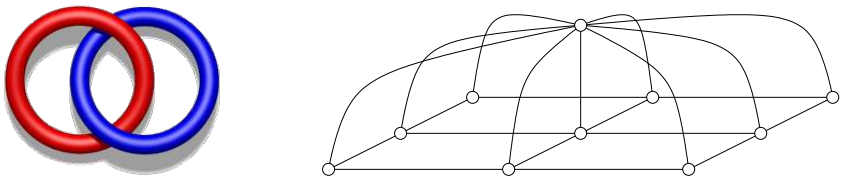


FIGURE 2.27 – Cycles entrelacés et plongement d'un graphe sans entrelacement. Tout graphe planaire augmenté d'un sommet, pas forcément connecté à tous les autres, possède un plongement sans entrelacement.

Par exemple, K_5 et $K_{3,3}$ possèdent un plongement sans entrelacement (il s'agit de graphes planaires augmentés d'un sommet, cf. figure 2.27) mais pas K_6 (voir le dessin de la figure 2.29 avec un seul entrelacement). [*Exercice.* Dessiner K_5 et $K_{3,3}$ sans entrelacement.] La preuve de l'impossibilité d'un tel plongement pour K_6 est loin d'être une simple remarque, et a fait l'objet de plusieurs articles complets [Sac81][Sac83][CG83].

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

21. On parle aussi de plongement dans $\mathbb{R}^3$. C'est toujours possible quelque soit le graphe même avec des segments de droites.

tester efficacement, disons en temps polynomial, qu'un plongement donné est sans entrelacement ?

En fait, un graphe est sans entrelacement [22] si et seulement si il ne contient aucun des septs graphes de la figure 2.28 comme mineurs [RST95]. Le théorème 2.8 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.

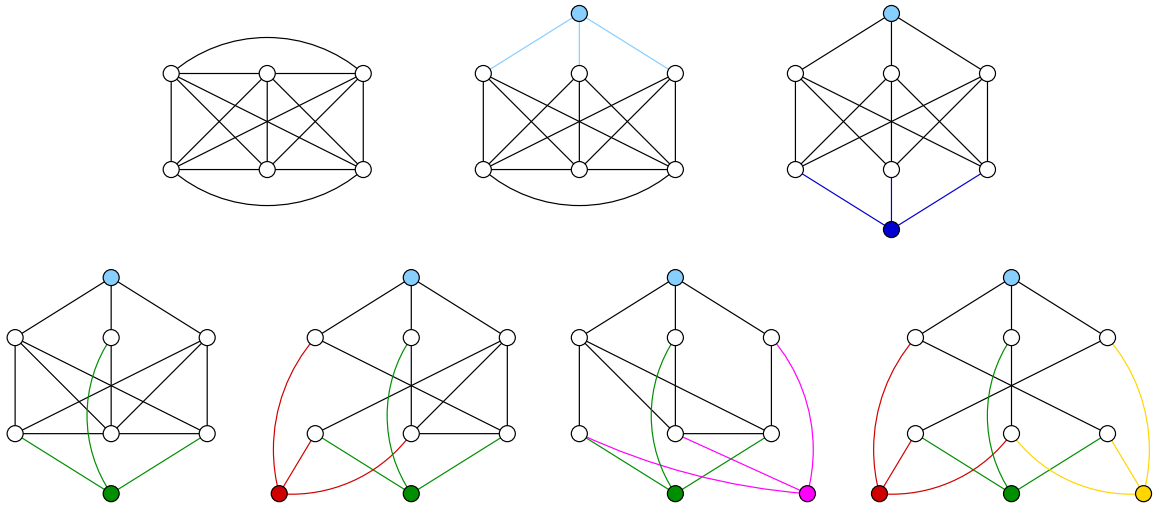


FIGURE 2.28 – Les mineurs exclus pour les graphes sans entrelacement.

En fait ces septs graphes, dont celui de Petersen, sont tous équivalents à K_6 par échanges Y-Δ (ou l'opération inverse Δ-Y) : on remplace un sommet ayant au moins trois voisins par un triangle connectant les trois sommets. La figure 2.29 montre une autre relation inattendue entre K_6 (en haut à droite) et le graphe de Petersen (en bas à gauche). Voir la figure 2.51 pour une autre représentation du 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 2.30 montre une courbe qui 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 $O(n^3)$.

De manière plus générale, en combinant les théorèmes 2.7 et 2.8 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 $\mathcal{F}$.

22. Notons que cette famille est close par mineurs : la contraction d'une arêtes 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.

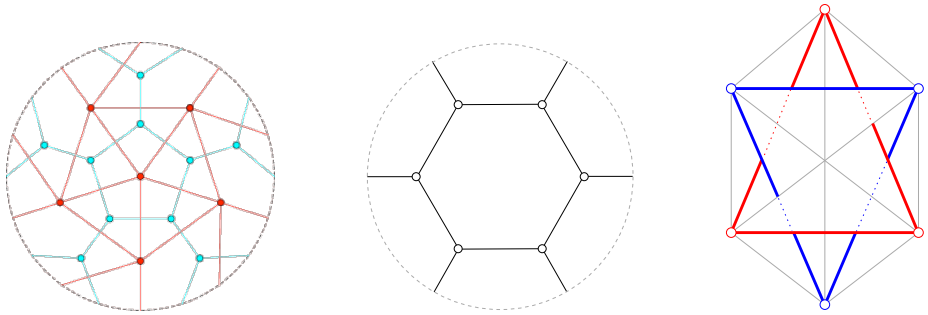


FIGURE 2.29 – Le graphe de Petersen et K_6 sont duals sur le plan projectif ; $K_{3,3}$ sur le plan projectif ; Un plongement de K_6 dans $\mathbb{R}^3$ avec un entrelacement.

FIGURE 2.30 – Plongement dans $\mathbb{R}^3$ d'un cycle avec un nœud.

Prenons un exemple. Soit $\mathcal{F}_k$ l'ensemble des graphes ayant un COUVERTURE PAR SOMMETS de taille au plus k. Résoudre COUVERTURE PAR SOMMETS de taille au plus k est, par définition, équivalent à savoir si un graphe donné appartient à $\mathcal{F}_k$. Or, la famille $\mathcal{F}_k$ est closes par mineurs. Donc, par le théorème 2.7, 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 COUPE-CYCLES de taille au plus k (*Feedback Vertex-Set* en Anglais). Il s'agit de déterminer si un graphe G possède un ensemble S de k sommets au plus 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 mineurs [23]. Ce problème est donc FPT en k.

Un autre exemple est COUVERTURE PAR FACES PLANAIRE de taille au plus k (*Planar Face Cover* en Anglais). Il s'agit de déterminer si un graphe planaire G possède dessin où les bords [24] d'au plus k faces couvrent tous les sommets de G. Cela peut permettre, par exemple, de contrôler des points d'intérêts à l'aide de k robots confinés aux régions. Les graphes planaires-extérieurs sont exactement les graphes ayant une couverture par faces de taille 1. Ce problème est NP-complet, et la famille de graphes associés est close par mineurs. [*Exercice*. Montrez que c'est effectivement le cas.] Ce problème est donc FPT en k.

23. La contraction d'arête ne peut faire apparaître de cycle.
24. Les bords ne sont pas forcément disjoints. Ils peuvent partager un ou plusieurs sommets ou arêtes.

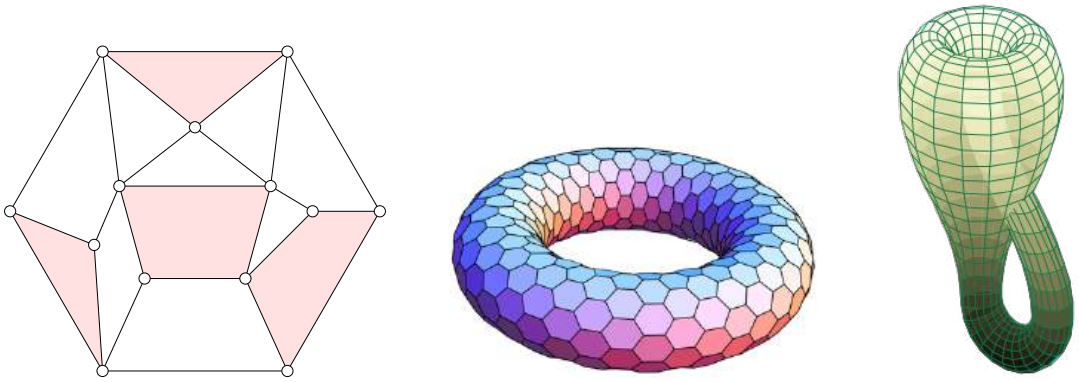


FIGURE 2.31 – Graphe planaire possédant une couverture par faces de taille 4 ; Graphe torique où toutes les faces sont des hexagones ; Quadrangulation d'une bouteille de Klein (surface non-orientable).

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 2.32). Notons aussi que cette technique produit des algorithmes non-uniformes, c'est-à-dire spécifiques pour chaque valeur du paramètre. [*Question*. Pourquoi ?]

FIGURE 2.32 – La suppression d'une arête 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, les obstructions pour les graphes dessinable (sans croisement d'arêtes) sur le plan projectif, comme $K_{3,3}$ sur la figure 2.29, sont au nombre de 103 [Arc81][GHW79]. Pour les graphes toriques (dessinables sur un tore comme sur la figure 2.31), ce nombre n'est même pas connu : c'est au plus 16 629 (en 2009, cf. [GMC09]). 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 très efficaces).

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, th. 16]. Cependant, il existe des algorithmes pour calculer les obstructions pour certaines familles comme les graphes de largeur arborescente k ou de genre k [AGK08].

2.10 Réduction aux graphes de *tree-width* bornée

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

Avant de donner un sens précis 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 [*Exercice.* Écrire la preuve de la validité.] :

Algorithme VC-forêt(T, k)

1. Si $E(T) = \varnothing$, 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($T \setminus \{v\}, k-1$).

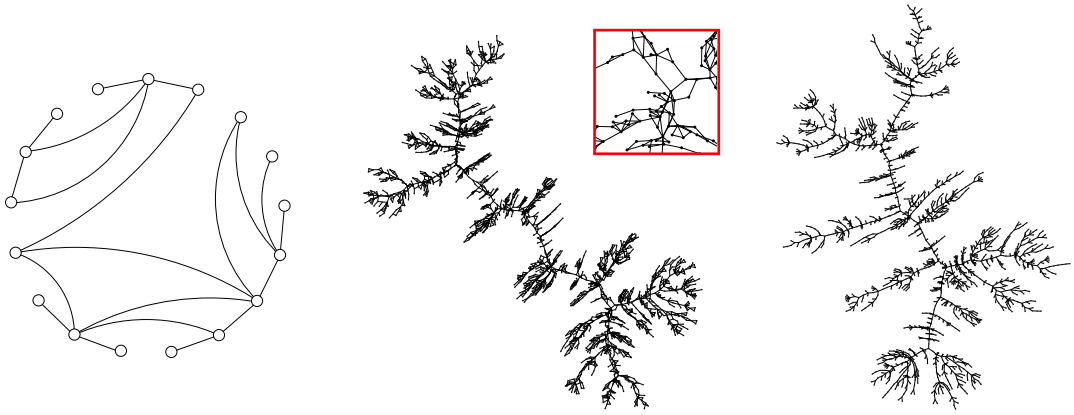
Cet algorithme travaille progressivement à partir des feuilles (et de leurs parents) en remontant vers la racine. Dans ce processus, il n'y a jamais de remise en question des décisions prises (*back tracking*), ce qui explique le temps polynomial. Ici on applique une règle simple de réduction de donnée (sommet de degré 1) et à la fin le noyau est vide !

L'espoir consiste donc à penser que ce genre d'algorithme continuera de fonctionner si le graphe d'entrée est suffisamment « proche » d'un arbre, comme l'illustre la figure 2.33. La largeur arborescence est un paramètre permettant de mesurer cette proximité. Plus elle est petite, plus le graphe ressemble à un arbre (pour ces derniers le paramètre vaut 1).

On va donc chercher à décomposer le graphe pour produire une « arborescence ». De manière très intuitive on résout le problème 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 dans les nœuds internes.

Pour Couverture par sommets dans un arbre, on peut résoudre le problème quasiment indépendamment sur des sous-arbres disjoints de l'arbre. Tant qu'il n'y a pas d'arête entre les sous-arbres, l'approche reste valide, et on peut ainsi considérer des sous-arbres de plus en plus grand. Le seul point délicat est lorsque les sous-arbres vont finir par se rejoindre en un ancêtre commun.

L'approche va en fait marcher pour de nombreux problèmes définis de manière « locale » comme Couverture par sommets, Ensemble indépendant, Ensemble dominant. Tant qu'il n'y a pas d'arêtes entre certaines parties du graphe (qui vont correspondre aux sous-arbres de la décomposition) on peut travailler de manière indépendante. Seule la



(a) (b) (c)

FIGURE 2.33 – (a) Un graphe planaire-extérieur peut-être dessiné sur le plan sans croisement d'arête de sorte que tous les sommets soient sur le bord de la face extérieure. (b) Un graphe planaire-extérieur aléatoire uniforme à 2 000 sommets (malheureusement dessiné de manière non-planaire comme le montre le zoom) faisant apparaître sa structure arborescente. Comme expliqué page 91, ce sont des graphes de largeur arborescente $\leqslant 2$. (c) Un arbre aléatoire uniforme à 2 000 sommets.

fusion nécessitera un travail spécifique.

2.10.1 Décomposition arborescente

Définition 2.6 (décomposition arborescente) *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. $\bigcup_{X \in V(T)} X = V(G)$;
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.*

Une propriété fondamentale des décompositions arborescentes est la suivante (voir figure 2.35) :

Propriété 2.1 *Soient $e = \{X, Y\}$ une arête de T, et A, B les deux arbres de la forêt $T \setminus \{e\}$. Alors, si dans G on supprime les sommets de $X \cap Y$, on déconnecte le sous-graphe induit par les sommets des sacs de A de celui induit par les sacs de B (les sommets de $X \cap Y$ étant exclus).*

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. Notons que cette propriété de séparation s'applique à tout sur-ensemble de $X \cap Y$, comme X ou Y. Cette propriété est évidemment la même chose pour

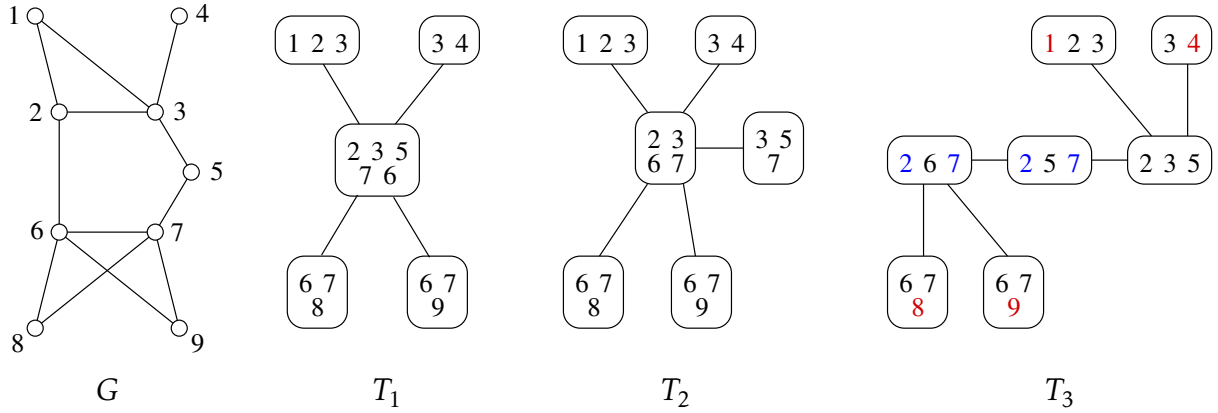


FIGURE 2.34 – Exemples de décompositions arborescentes d'un graphe G. Dans la décomposition T_3, l'arête $\{2,6,7\}-\{2,5,7\}$ avec son intersection $\{2,7\}$ déconnecte dans G les sommets 1 et 4 des sommets 8 et 9.

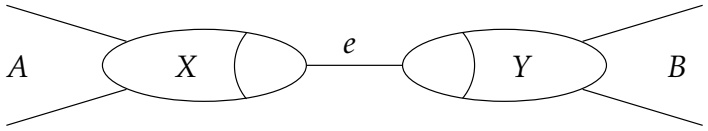


FIGURE 2.35 – Les sommets de $X \cap Y$ séparent A de B.

l'arbre T à savoir : tout chemin d'un nœud de A vers un nœud de B passe par l'arête e (et aussi par le nœud X et le nœud Y). Donc la « structure » du graphe se décrit plus ou moins finement par celle de l'arbre.

Preuve de la propriété 2.1. Soit V_A (resp. V_B) l'ensemble des sommets de $G \setminus (X \cap Y)$ contenus dans les sacs de A (resp. de B). Rappelons que par la 1ère propriété de couverture des sommets, tout sommet de G est dans un sac de T, et donc dans un sac de A, de B ou des deux à la fois.

Par contradiction, supposons qu'il existe une arête $\{u,v\}$ de G connectant un sommet $u \in V_A$ à un sommet $v \in V_B$.

Montrons d'abord que $V_A \cap V_B = \varnothing$. Soit U l'ensemble des sacs contenant u. Par le choix de u, U contient au moins un sac de l'arbre A. Si U contient aussi un sac de B, alors, comme U est un arbre (2e propriété), il doit contenir les sacs X et Y. Et donc $u \in X \cap Y$ ce qui n'est pas possible. Il suit que $u \notin V_B$.

Ainsi, les sacs contenant $u \in V_A$ sont tous inclus dans A. De même les sacs contenant $u \in V_B$ sont tous inclus dans B. Donc les sacs contenant u et v doivent être dans l'intersection $A \cap B$ qui est malheureusement vide par construction de A et B : c'est en contradiction avec la 2e propriété de couverture des arêtes de G.

En conséquence, l'arête $\{u,v\}$ entre V_A et V_B n'existe pas ce qui démontre la propriété 2.1. □

Bien sûr il y a plusieurs décompositions arborescentes possibles pour un graphe. Il y en a toujours une triviale composée d'un arbre à un seul nœud et contenant tout le graphe. Bien que valide, ce n'est pas la plus intéressante des décompositions, cet arbre trivial ne permettant pas de dire quoique ce soit sur la structure du graphe, la propriété 2.1 n'étant ici d'aucune utilité. [*Exercice.* Montrez que la décomposition en composantes bi-connexes (qui se joignent par les sommets d'articulations) forme aussi une décomposition arborescente.]

Nous allons nous intéresser aux décompositions où les sacs sont les plus petits possibles, c'est-à-dire qui contiennent le moins de sommets possibles.

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

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

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 2.36 donne un exemple de décomposition arborescente d'un arbre. Une telle décomposition peut se définir récursivement en enlevant les feuilles jusqu'à la dernière arête. Cela correspond aussi à l'heuristique du degré minimum décrite page 90. [*Exercice.* Montrez que tout arbre possède une décomposition arborescente isomorphe à l'arbre lui-même.]

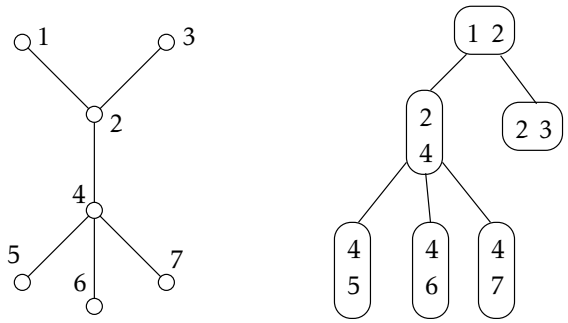


FIGURE 2.36 – Décomposition arborescente de largeur 1 pour un arbre.

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 s'il n'y a pas d'inclusion de sacs voisins — ce qui, le cas échéant, peut être detecté et corrigé facilement en fusionnant les sacs ou en contractant les arêtes — alors la décomposition possède au plus n sacs. On parle alors de décomposition « réduite ». Dans toute la suite, on supposera toujours que les décompositions sont réduites.

Il existe d'autres définitions pour la largeur arborescente d'un graphe. Une relativement simple dit que $\mathrm{tw}(G) = k$ si et seulement si G est un sous-graphe couvrant d'un k-arbre. Un *k-arbre* est un graphe qui peut être obtenu itérativement en ajoutant chaque nouveau sommet que l'on connecte à une clique de taille k choisie dans le graphe (qui au départ est une clique taille k). Les k-arbres sont aussi des graphes sans cycle induit de longueur supérieure à trois. On appelle aussi ces derniers graphes *triangulés* ou graphes *cordaux*. Une *triangulation* de G est un graphe obtenu à partir de G en ajoutant des arêtes jusqu'à un graphe triangulé. La largeur arborescente de G peut aussi être définie comme la taille de la clique maximum moins minimisé sur toutes les triangulations de G. Arbres, cliques et graphes par intervalles [25] par exemple sont des graphes cordaux. Et leurs largeurs arborescentes est donc la taille de leur clique maximum moins un.

2.10.2 Calcul de décomposition arborescente

Largeur arborescente (majorant) de quelques graphes à n sommets :

- graphes sans arête 0
- arbres et forêts 1
- graphes complets (cliques) ..... $n-1$
- cycles, graphes planaires-extérieurs ou séries-parallèles 2
- les 16 430 graphes de la base moléculaires KEGG [26] 4
- graphes de flot de contrôle d'un programme [27] `C` sans `goto` [Tho98] 6
- graphes dont le plus long chemin est de longueur k ..... k
- graphes ayant un parcours DFS de profondeur k ..... k
- graphes mineurs d'un graphe [28] de largeur arborescente k ..... k
- graphes ayant une couverture par sommets [29] de taille k ..... k
- graphes ayant un coupe-cycles [30] de taille k ..... $k+1$
- graphes qui sont mineurs d'une grille $p \times q$ ..... $\min\{p,q\}$
- graphes qui excluent une grille $k \times k$ comme mineur [31] [CT19] ..... $k^{9+o(1)}$

25. Graphes d'intersections d'intervalles d'une ligne : les sommets sont des intervalles et deux sommets sont voisins ssi leurs intervalles s'intersectent.

26. Plus précisément, il s'agit de la base KEGG COMPOUND où seuls deux graphes sont de largeur arborescente 4 et 77% sont de largeur 2, voir [YAK14, Table 16.1].

27. Voir aussi [GMT02] pour les programmes Java où 73% des 604 méthodes (en 2002) du packages de `java.lang` ont une largeur arborescente 3 ou moins. Une seule méthode des miliers testées atteint une largeur de 5 :`receive(DatagramPacket)` de la classe `java.net.DataSocket`.

28. La famille des graphes de largeur arborescente au plus k est close par mineurs. Donc la largeur arborescente de tout mineur d'un graphe G (en particulier d'un sous-graphe de G) est au plus celle de G.

29. La décomposition arborescente est une étoile, un arbre de profondeur 1, dont le sac au centre est C et les feuilles composées des voisinages fermés $N[u]$ pour tous les sommets $u \notin C$.

30. Voir page 79 pour la définition. La décomposition arborescente est composée de la décomposition arborescente de la forêt $G \setminus S$, S étant le coupe-cycles de G, où l'on ajoute S à chacun de ses sacs.

31. Cf. aussi l'équation (2.2) de la section 2.10.6.

- graphes planaires ayant une couverture par faces [32] de taille k . . . . $9\sqrt{3k/5}-1$
- graphes k-séparables [33] [DN19] . . . . . $15k$
- graphes planaires . . . . . $2\sqrt{3n}-1$
- graphes de genre eulérien [34] g [DMW17] . . . . . $2\sqrt{(2g+3)n}-1$
- graphes k-planaires [35] [DEW15] . . . . . $2\sqrt{6(k+1)n}-1$
- graphes qui excluent un mineur à k sommets . . . . . $O(k\sqrt{n})$

L'intuition du résultat en $O(\sqrt{n})$ pour la largeur arborescente des graphes planaires, qui est un résultat important, est la suivante. Tout d'abord, à cause du résultat sur les graphes k-séparables, il suffit d'arriver à peu près à « couper en deux » le graphe, c'est-à-dire de trouver un *séparateur équilibré* [33], en supprimant $k = O(\sqrt{n})$ sommets.

Si le graphe est une grille $p \times q$, avec ou sans diagonale dans chaque face carrée, un graphe planaire certes particulier mais assez représentatif, alors une ligne ou une colonne fait très bien l'affaire puisque $\min\{p,q\} \leqslant \sqrt{pq}$. [*Question*. Pourquoi ?] Ensuite, et c'est plus intuitif, si le graphe est la triangulation de Delaunay d'un ensemble de n points aléatoires, alors le nombre de sommets de la triangulation, n, peut être vu comme la mesure de la « surface du graphe ». Et on a l'impression qu'on peut toujours couper en deux cette surface (et donc ce graphe), selon une coupe de longueur le diamètre, comme le montre la figure 2.37. Rappelons que dans un disque de diamètre $2r$ la surface est πr^2.

On verra dans la proposition 2.4 que tout graphe planaire de rayon r, c'est-à-dire qui possède un arbre couvrant de profondeur r, est de largeur arborescente au plus $3r$, ce qui implique un séparateur équilibré de taille $3r$. [*Exercice**. Montrez que tout graphe ayant une décomposition arborescente de largeur ω possède un séparateur équilibré de taille $\omega+1$.]

Le problème Tree-Width est défini comme suit et est malheureusement NP-complet [ACP87] :

---

32. Voir page 79 pour la définition. En combinant le fait que $\mathrm{bw}(G) \leqslant 6\sqrt{3k/5}$ [KT11, th. 3] où $\mathrm{bw}(G)$ est un paramètre de G appelé *branch-width* et que $\mathrm{tw}(G)+1 \leqslant \max\{3\mathrm{bw}(G)/2, 2\}$ [RS91]. Voir aussi [HW17].

33. Graphes dont chaque sous-graphe possède un *séparateur équilibré* de taille k, c'est-à-dire un ensemble de k sommets dont la suppression le déconnecte en deux parties d'au plus 2/3 du nombre total de sommets. D'autres variantes sur la notion de séparateur existent et sont toutes reliées à la largeur arborescente, voir [HW17].

34. C'est une généralisation du genre classique comme défini au paragraphe 2.4. Elle considère non seulement les surfaces orientables, comme le genre classique, mais aussi non-orientables. Le genre eulérien d'une surface est $2h+c$ si elle peut-être obtenue à partir d'une sphère en ajoutant h anses et c *cross-caps*. Une *cross-cap* est un bord de la surface sur lequel est attaché un ruban de Möbius et où les points antipodaux correspondent. Cela permet de représenter les graphes comme celui le plus à droite de la figure 2.38.

35. Graphe qui peut-être dessiné dans le plan où chaque arête coupe au plus k autres. Les graphes planaires sont donc 0-planaires.

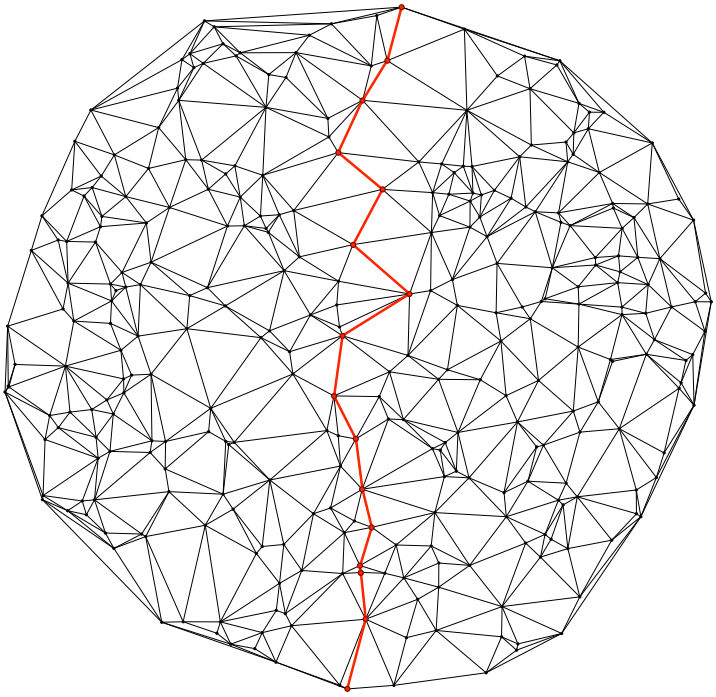


FIGURE 2.37 – Triangulation de Delaunay sur $n = 314 \approx \pi 10^2 = \pi r^2$ points aléatoires uniformes sur un disque, et un ensemble de $16 < 2r$ sommets coupant en deux sous-graphes avec chacun $149 < n/2$ sommets.

TREE-WIDTH

Instance: un graphe G et un entier k
Question: $\mathrm{tw}(G) \leqslant k$?

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 2.9). [*Exercice.* Vérifiez que tel est bien le cas.] 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 2.38. Pour $k = 4$, la liste n'est pas connue mais il y en a au moins [36] 76. Pour $k \geqslant 5$, des algorithmes permettent en principe de calculer ces listes, cf. [LA91][AGK08][Luc07][SST20]. Pour fixer les idées, le nombre de mineurs interdits pour les graphes de *path-width* k, une variante de la largeur arborescente, est au moins $(k!)^2$, chacun de taille au moins 3^k [TUK94].

En fait, il existe un algorithme en temps $2^{O(\mathrm{tw}(G)^3)} \cdot n$ donnant une décomposition arborescente de largeur optimale : c'est donc un algorithme linéaire pour une largeur

36. La thèse présentée [San93] donne 75 en dehors de K_6 qui doit être rajouté à cette liste.

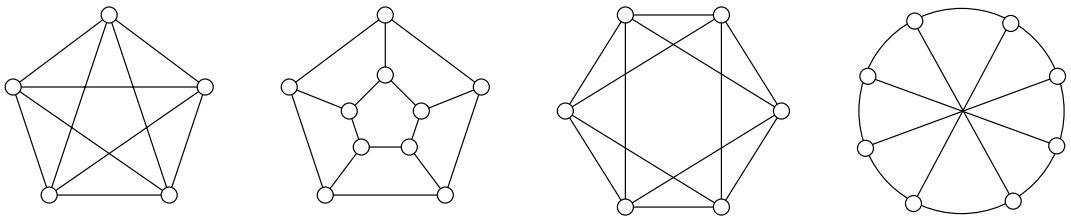


FIGURE 2.38 – Les mineurs exclus pour les graphes de largeur arborescente $\leqslant 3$. En fait, il existe un algorithme linéaire pour reconnaître ces graphes [AP87] permettant de se passer du test de mineur du théorème 2.8.

arborescente fixée. Comme on va le voir dans la suite, une décomposition arborescente de largeur en $O(\mathrm{tw}(G))$ au lieu de $\mathrm{tw}(G)$ exactement est souvent suffisante. De telles décompositions peuvent être calculées en temps certes super-linéaire, mais surtout simplement exponentiel en $\mathrm{tw}(G)$ et non pas exponentiel en $\mathrm{tw}(G)^3$.

Pour être plus précis, en temps $2^{O(k)} \cdot n$, soit l'algorithme répond que $\mathrm{tw}(G) > k$, ou bien renvoie une décomposition arborescente pour G de largeur $O(k)$. C'est le même principe pour les résultats présentés dans le théorème 2.9 ci-dessous, qui peuvent différer dans la complexité en temps (fonction de k et n), la largeur (fonction de k) de la décomposition produite, et leur facilité d'implémentation.

Évidemment, les constantes cachées dans les notations « O » jouent un rôle essentielle en pratique. Il existe de très nombreux algorithmes calculant ou approximant $\mathrm{tw}(G)$, dont certains très récents, soulignant l'intérêt de ce paramètre. Les résultats suivants (qui sont complexes et que nous admettrons dans ce cours [37]) sont classés selon la largeur des décompositions produites. Dans ce cours nous utiliserons le dernier en date, l'un des rare à fournir en temps linéaire et simplement exponentiel en k une décomposition arborescente de largeur optimale à un facteur constant près.

Théorème 2.9 *Pour tout graphe G à n sommets et entier k, il est possible de déterminer si $\mathrm{tw}(G) > k$, et sinon de construire une décomposition arborescente de largeur $\leqslant \omega(k)$ en temps $f(k) \cdot \mathrm{poly}(n)$ où les paramètres sont les suivants :*

37. Tous sont basés sur la recherche d'un séparateur équilibré [33] de taille $O(k)$.

$\omega(k)$	$f(k)$	poly(n)	références
k	1	n^{k+2}	[ACP87]
k	1	1.7347^n	[FTV15]
k	$2^{O(k^2)}$	n^4	[KL23]
k	$2^{24k^3+O(k^2)}$	n	[Bod96]
$(1+\varepsilon)k$	$k^{O(k/\varepsilon)}$	n^4	[KL23]
$2k+1$	$2^{O(k)}$	n	[Kor22]
$3k+2$	$3^{3k}k^2$	$n\log n$	[Ree92][38]
$3k+4$	$2^{O(k)}$	$n\log n$	[BDD+16]
$3.\bar{6}k$	$2^{3.7k}k^3$	$n^3\log^4 n$	[Ami01][39]
$4k$	$2^{4.38k}k$	n^2	[Ami01]
$4k+3$	3^{3k}	n^2	[RS95]
$4.5k$	$2^{3k}k^{3/2}$	n^2	[Ami01]
$5k+4$	$2^{O(k)}$	n	[BDD+16]
$5k+4$	$2^{6.76k}$	$n\log n$	[BF21b][BF21a]
$5k+4$	$k!2^{24k}$	$n\log n$	[Ree92][40]
$6k+5$	$2^{O(k\log k)}$	$n\log^2 n$	[MT91][41]
$8k+7$	$2^{O(k\log k)}$	$n\log^2 n$	[Lag96]
$O(k\sqrt{\log k})$	1	$n^{O(1)}$	[FHL05]
$O(k\log k)$	$k\log k$	n^4	[Ami01]
$O(k\log k)$	$k^5\log k$	$n^3\log^4 n$	[Ami01]
$80k\ln k+560k$	$k^{O(1)}$	$n^{O(1)}$	[BKMT04]
$O(k^2)$	k^7	$n\log n$	[FLS+18]

Ces algorithmes [42] permettent d'approximer $\mathrm{tw}(G)$ à un facteur $\omega(\mathrm{tw}(G))/\mathrm{tw}(G)$ près, simplement en les testant pour $k = 1, 2, 3, 4, 5, \ldots$ jusqu'à obtenir une réponse positive, disons pour $k = k_0$. On en déduit alors une décomposition de largeur au plus $\omega(k_0) \leqslant \omega(\mathrm{tw}(G))$ car bien évidemment $k_0 \leqslant \mathrm{tw}(G)$ et les fonctions $\omega(k)$ sont croissantes. La décomposition est construite en un temps augmenté d'un facteur $k_0 \leqslant \mathrm{tw}(G)$ par rapport à l'algorithme utilisé.

On peut faire un peu plus rapide en « sondant » par dichotomie la plus petite valeur $k_0 \leqslant \mathrm{tw}(G)$ de k qui donne une réponse positive. On teste d'abord $k = 1, 2, 4, 8, 16, \ldots$. On

38. L'article original donne $\omega(k) \leqslant 5k$ mais une modification de [Bod93, th. 6.1] permet d'en déduire $\omega(k) \leqslant 3k+2$.

39. Article publié dans une conférence d'IA et recherches financées par la DARPA.

40. L'article original donnait $\omega(k) \leqslant 8k+7$, mais cette nouvelle analyse due à [BF21b] donne une meilleure borne.

41. Algorithme de type Monté Carlo : avec une probabilité $p > 0$ fixée il peut se tromper en répondant que $\mathrm{tw}(G) > k$.

42. Historiquement, il y en a beaucoup d'autres (voir aussi [KL23, table 1]), comme celui de Robertson et Seymour [RS95] avec $\omega(k) = k$ et un temps de $f(k)n^2$ pour une fonction f indéterminée qui, au moment de la publication, n'était pas connue pour être calculable. Il y a aussi celui de [MT91] qui est si $\mathrm{tw}(G) \leqslant 3$.

en déduit alors un intervalle $]2^{i-1}, 2^i]$ contenant k_0 après seulement $i = \lceil \log k_0 \rceil$ tests. Cet intervalle est de longueur au plus k_0. On peut alors effectuer une dichotomie classique pour trouver la valeur k_0 recherchée, donc avec encore $\lceil \log k_0 \rceil$ tests suplémentaires. Au final, le temps de construction est seulement multiplié par un facteur au plus $2\lceil \log k_0 \rceil = O(\log \mathrm{tw}(G))$ au lieu de $O(\mathrm{tw}(G))$.

Calculer une décomposition arborescente optimale (cf. lignes 1-2 du théorème 2.9) pour de grands graphes de largeur arborescente $\geqslant 10$ en moins de 24h reste un challenge (voir [DK07][ZH09][BK10][BK11] pour des études expérimentales). Le meilleur algorithme (non FPT) calculant la largeur arborescente d'un graphe à n sommets est en $O(1.8899^n)$ [FTKV08]. Pour les graphes planaires, il existe des algorithmes polynomiaux comme [KT16] donnant une approximation constante en temps linéaire.

Il existe cependant des heuristiques, sans aucune garantie, mais très simples à calculer. En voici deux :

1. **L'heuristique du parcours en profondeur.** On calcule un arbre T enraciné et couvrant le graphe G selon un parcours en profondeur. On forme un sac pour chaque branche de l'arbre, le sac S_i étant celui de la i-ème branche, c'est-à-dire le chemin allant de la i-ème feuille de T selon le parcours à la racine. L'arbre de décomposition est alors simplement le chemin $S_1 - S_2 - \cdots - S_k$ où k est le nombre de feuilles de T.
2. **L'heuristique du degré minimum.** On effectue une simple récurrence comme ceci. Si G est une clique, l'arbre possède un seul sac contenant cette clique, et on s'arrête. Sinon, on choisit un sommet u de degré minimum dans G, et on forme un sac X composé de u et de ses voisins dans G. On construit alors récursivement une décomposition arborescente T d'un nouveau graphe obtenu à partir de $G \setminus \{u\}$ dans lequel on a rajouté toutes les arêtes entre les voisins de u. Les voisins de u forment donc une clique dans ce nouveau graphe. Au final, on renvoie l'arbre formé de T et du sac X qui est connecté à un sac de T contenant tous les voisins de u de G. On peut vérifier que toute décomposition arborescente d'un graphe contenant une clique possède au moins un sac la contenant.

Notez que les graphes k-dégénérés (voir la définition 2.2), que l'on peut éplucher par la suppression d'un sommet de degré au plus k, ne sont pas nécessairement de largeur arborescente bornée par k ou même une fonction de k — même si c'est vrai pour $k = 0$ ou 1. Par exemple, les grilles $p \times q$ qui sont 2-dégénérées, sont de largeur arborescente $\min\{p, q\}$.

Une autre heuristique, pas forcément meilleure que les précédentes, consiste à calculer une couverture par sommets C de petite taille (on verra au chapitre 3 qu'on peut facilement approcher celle de taille minimum). Comme expliqué précédemment dans la note en bas de page [29], une étoile avec comme sacs C et $N[u]$ pour tout $u \notin C$ donne une décomposition arborescente.

[*Exercice*. Vérifier qu'il s'agit dans chacun des cas d'une décomposition arborescente

valide. Donner un exemple de graphe de rayon h et dont l'heuristique du DFS donne des sacs de taille beaucoup plus grande que h.]

L'heuristique du degré minimum fonctionne très bien pour les graphes planaires-extérieurs. En effet, il existe toujours un sommet de degré au plus deux. Et si on le supprime en ajoutant une clique entre ses voisins on obtient encore un graphe planaire-extérieur. Cela montre que ces graphes sont de largeur arborescente au plus deux.

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, $G[T_X] := G[\cup_{Y \in V(T_X)} Y]$.

2.10.3 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 planaire. Notez que le problème de décision de la k-coloration des graphes planaires est triviale pour $k = 1, 2, 4$. [*Question*. Pourquoi ?]

Théorème 2.10 *Étant donnée une décomposition arborescente de largeur ω d'un graphe G à n sommets, on peut résoudre* 3-Coloration *pour G en temps $2^{O(\omega)} \cdot n$.*

Grâce au théorème 2.9, on peut calculer pour G une décomposition arborescente de largeur $\omega \leqslant 2\text{tw}(G) + 1$ en temps $2^{O(\text{tw}(G))} \cdot n$. On en déduit que 3-Coloration peut être résolu en $2^{O(\text{tw}(G))} \cdot n$, ce qui en fait un problème FPT en la largeur arborescente du graphe.

Preuve du théorème 2.10. Soit T la décomposition arborescente de G de largeur ω, et on note R sa racine, un sac quelconque de T. Dans la suite on notera $\text{COL}(X)$ l'ensemble des 3-colorations possibles du sous-graphe $G[X]$.

Comme souvent en programmation dynamique, on est amené à calculer récursivement une certaine variable que l'on va définir. Et pour que la récurrence fonctionnement tout en résolvant notre problème, il faut demander un peu plus qu'une simple 3-coloration pour chaque sac.

Pour chaque sac X, on va calculer récursivement un ensemble de 3-colorations comme suit (cf. figure 2.39 pour une illustration) :

- SOL(X) est l'ensemble des 3-colorations de $G[X]$ qu'il est possible d'*étendre* à une 3-coloration du sous-graphe $G[T_X]$.

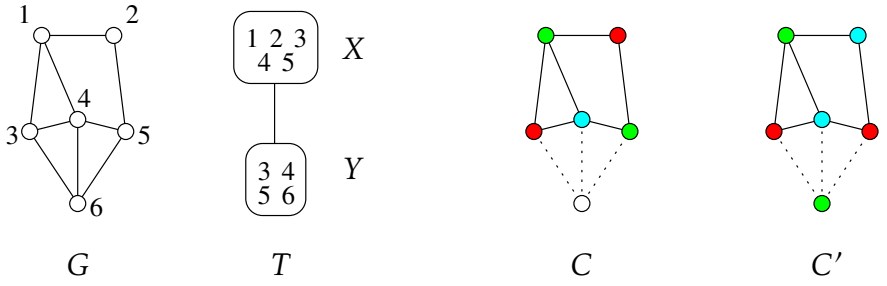


FIGURE 2.39 – Une décomposition arborescente T pour G. Une 3-coloration C de $G[X]$ qui n'est pas dans SOL(X). Et une autre 3-coloration $C' \in$ SOL(X).

Une coloration de X est une fonction $C : X \to \{1,2,3\}$ 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 ordonnés, étiquetés de manière unique de 1 à n). Donc COL(X) peut être vu comme un ensemble de suites de $|X| \leqslant \omega + 1$ entiers de $\{1,2,3\}$. On a bien évidemment que $|\text{SOL}(X)| \leqslant |\text{COL}(X)| \leqslant 3^{|X|} \leqslant 3^{\omega+1}$, car SOL($X$) $\subseteq$ COL(X).

Bien sûr, G est 3-coloriable si et seulement si SOL(R) $\neq \varnothing$, car $G[T_R] = G[T] = G$. On cherche donc à calculer SOL(R).

Remarquons que si X est une feuille, alors $G[T_X] = G[X]$ et donc SOL(X) = 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 $|X|^2$ arêtes [43]. Donc pour chaque feuille X, SOL(X) se calcule en un temps [44] $O(3^{|X|} \cdot |X|^2) = 2^{O(\omega)}$.

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

On peut vérifier que $C \in$ SOL(X) si et seulement si elle est compatible avec tous ses fils. En effet, si $C \in$ SOL(X), alors, par définition de SOL(X), C est une coloration de

43. En fait, si G est 3-coloriable $G[X]$ ne peut avoir plus de $|X|^2/3$ arêtes (sinon $G[X]$ n'est pas 3-coloriable), nombre qui est atteint pour un graphe tri-parties complet $K_{n/3,n/3,n/3}$.

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

$G[T_X]$ et donc de $G[T_{Y_i}]$. Il doit exister des colorations $C_i \in \mathrm{SOL}(Y_i)$ 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 $C \in \mathrm{COL}(X)$ est compatible avec tous ses fils, alors $C \in \mathrm{SOL}(X)$ car, d'après la propriété 2.1, pour chaque i, il ne peut avoir d'arête ni entre $Y_i \setminus X$ et $X \setminus Y_i$, ni entre $Y_i \setminus X$ et $Y_j \setminus X$. (Cf. figure 2.40.) Donc C est bien une coloration de $G[T_X]$.

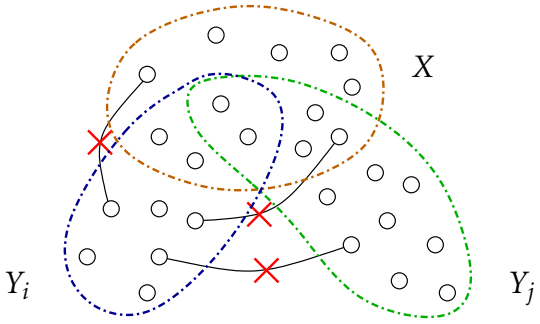


FIGURE 2.40 – D'après la propriété 2.1, il n'y a pas d'arête ni entre $Y_i \setminus X$ et $X \setminus Y_i$, ni entre $Y_i \setminus X$ et $Y_j \setminus X$. Les arêtes entres Y_i et Y_j, si elles existent, doivent être dans X.

En fait, s'il y a une arête entre des sacs fils Y_i et Y_j alors elle est dans X (cf. figure 2.40). C'est cette dernière propriété qui permet de calculer indépendamment $\mathrm{SOL}(Y_i)$ et $\mathrm{SOL}(Y_j)$, les contraintes liées aux arêtes n'intervenant que lors de la fusion dans le sac X.

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

Algorithme $\mathrm{SOL}(X)$

---

1. Calculer $S := \mathrm{COL}(X)$.
2. Pour chaque fils Y_i de X faire :

   Pour toutes les colorations $C \in \mathrm{COL}(X)$ et $C_i \in \mathrm{SOL}(Y_i)$, s'il existe $u \in X \cap Y_i$ tel que $C(u) \neq C_i(u)$, alors $S := S \setminus \{C\}$.
3. Renvoyer S.

Bien sûr, en pratique on ne procèdera pas comme ceci. On commencera plutôt à calculer une 3-coloration de $G[X]$ et au fur et à mesure on testera la compatibilité avec tous les fils, passant à la coloration suivante (rejet) dès qu'une contrainte est violée.

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 $O(|X| + |Y_i|)$, 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 tri est déjà faite, l'algorithme de calcul de $\mathrm{SOL}(X)$ a une

complexité de :

$$2^{O(\omega)} + \sum_{i=1}^{t} \left(O(\underbrace{|X| + |Y_i|}_{2(\omega+1)}) + \underbrace{|\mathrm{COL}(X)|}_{3^{\omega+1}} \cdot \underbrace{|\mathrm{SOL}(Y_i)|}_{3^{\omega+1}} \cdot O(\underbrace{|X \cap Y_i|}_{\omega+1}) \right) \leqslant \deg_T(X) \cdot 2^{O(\omega)} .$$

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

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

□

[*Exercice*. Dans l'Algorithme $\mathrm{SOL}(X)$ ci-dessus, donnez une implémentation plus efficace pour tester la compatibilité entre X et chaque fils Y_i.]

[*Exercice*. Comment calculer la 3-coloration si elle existe ?]

2.10.4 Ensemble dominant pour *tree-width* bornée

On va maintenant considérer un problème avec un paramètre : Ensemble dominant de taille $\leqslant k$. Comme précédemment, l'algorithme consiste à d'abord calculer une décomposition arborescente T de G de largeur $\omega \leqslant 2\mathrm{tw}(G) + 1$ en temps $2^{O(\mathrm{tw}(G))} \cdot n$, puis de résoudre le problème directement sur T en temps $2^{O(\omega)} \cdot n$, ce qui fait un total de $2^{O(\mathrm{tw}(G))} \cdot n$. Le problème est donc FPT en la largeur arborescente du graphe, alors qu'on pense qu'il ne l'est pas en la taille de l'ensemble.

Théorème 2.11 ([vRBR09]) *Étant donnée une décomposition arborescente de largeur ω d'un graphe G à n sommets, on peut résoudre Ensemble dominant de taille $\leqslant k$ pour G en temps $O(3^\omega \omega^2 \cdot n)$.*

Preuve. Soit T une décomposition arborescente de G de largeur ω, et R un sac de T choisi comme racine. On notera $\mathrm{Père}(X)$ le sac parent de X dans T, avec $\mathrm{Père}(R) = \varnothing$.

Naïvement, on aurait envie de faire comme pour la coloration : (1) on suppose qu'on a un ensemble dominant pour chaque sous-graphes fils $G[T_{Y_i}]$ de taille disons t_i ; (2) on calcule tous les ensembles dominants pour $G[X]$; et (3) on ne garde que les ensembles dominants « compatibles » avec les fils, en particulier ceux dont la taille n'excèdent pas k (en tenant compte des tailles t_i).

Si on trouve une solution pour R, alors G possède un ensemble dominant de taille $\leqslant k$. Cependant, le contraire n'est pas vrai. On risque par cette méthode de rater des ensembles dominants. Car pour construire en ensemble dominant pour $G[T_X]$ on a pas

forcément besoin d'avoir un ensemble dominant complet pour chacun des fils $G[T_{Y_i}]$. Certains sommets de $Y_i \cap X$ pourraient ne pas a être couverts par des sommets de $G[T_{Y_i}]$ s'ils le sont déjà par des sommets de $X \setminus Y_i$ comme illustré par la figure 2.41.

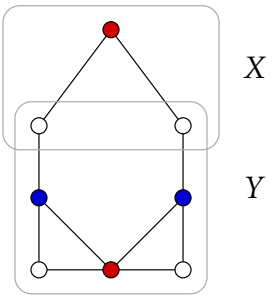


FIGURE 2.41 – $G[X \cup Y]$ a-t'il un ensemble dominant de taille $k = 2$? Deux sommets de Y (en bleu) sont nécessaires pour dominer $G[Y]$, et un seul pour $G[X]$. Mais aucune solution de taille k pour $G[Y]$ permet de dominer aussi $G[X]$. En procédant naïvement, on ne trouvera pas de solution de taille 2 (ici en rouge) pour $G[X \cup Y]$.

Pour chaque sac X, on calcule des feuilles à la racine l'ensemble des triplets suivants (cf. la figure 2.42) :

$$\begin{aligned} \mathrm{DOM}(X) := \{\, (S,P,t) : \exists W \subseteq V(G[T_X]) \text{ dominant } V(G[T_X]) \setminus \mathrm{PÈRE}(X), \\ S = W \cap X, \\ P = \mathrm{PÈRE}(X) \setminus N[W], \\ t = |W| \leqslant k \,\}\,. \end{aligned}$$

On rappelle que $N[W]$ représente les sommets de W et leurs voisins (voisinages fermés). Par conséquent P représente les sommets de $G[T_X]$ non dominés par W. Ils sont forcément dans $\mathrm{PÈRE}(X)$. Les sommets de P sont l'analogue des sommets marqués dans le problème ENSEMBLE DOMINANT ANNOTÉ qui, comme expliqué au paragraphe 2.7.3, se révèle être une variante incontournable pour résoudre ENSEMBLE DOMINANT.

Il est clair que $\mathrm{DOM}(R) \neq 0$ si et seulement si G possède un ensemble dominant de taille $\leqslant k$. Il suffit de réécrire la définition de $\mathrm{DOM}(R)$ avec $\mathrm{PÈRE}(R) = \varnothing$.

Pour l'implémentation, il faut lister tous les mots possibles sur trois lettres codant chacun un état possible d'un sommet de X qui est : soit dans S, soit dans P, soit dans l'ensemble restant $X \setminus (S \cup P)$ (cf. les trois couleurs des sommets de X sur la figure 2.42).

Pour une feuille X, on peut donc facilement calculer $\mathrm{DOM}(X)$: (1) en listant tous les mots de $|X|$ lettres ; (2) en construisant les ensembles S et P correspondant ; et (3) en vérifiant chacune des conditions de $\mathrm{DOM}(X)$. Cela prend un temps $O(3^{|X|} \cdot |X|^2) = O(3^\omega \cdot \omega^2)$ puisque $G[X]$ possède au plus $|X|^2$ arêtes.

Pour les autres sacs, il faut définir la règle de fusion.

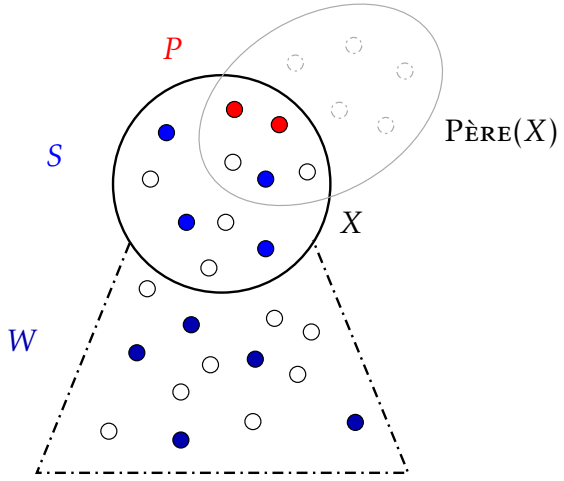


Figure 2.42 – Illustration de la définition de DOM(X). Les sommets de W sont tous les sommets bleutés (en bleu foncé ou bleu clair), et $S = W \cap X$. Les sommets de P (en rouge), n'étant pas dominés par de W, doivent être dominés par des sommets pris dans $\text{Père}(X) \setminus X$.

Règle de fusion. Soit $Y_1, \ldots, Y_d$ les d fils de X. Alors $(S, P, t) \in \text{DOM}(X)$ si et seulement si $\exists S \subseteq X$, $\exists P \subseteq \text{Père}(X)$ et $\forall i, \exists (S_i, P_i, t_i) \in \text{DOM}(Y_i)$ tels que :

1. $S \supseteq S_i$ domine $((X \setminus Y_i) \cup P_i) \setminus \text{Père}(X)$;
2. $P \supseteq P_i \setminus N[S]$; et
3. $t = |S| + \sum_i (t_i - |S_i \cap X|) \leqslant k$.

[*Cyril*. Preuve à finir.]

Encore une fois, en pratique on rejettera un triplet (S, P, t) candidat pour DOM(X) dès qu'une condition est fausse. Le challenge est d'obtenir une complexité globale en $O(3^\omega \omega^2 \cdot n)$, soit $O(3^\omega \omega^2)$ par arête de T. On ne donnera pas ici les détails disponibles dans [vRBR09]. □

[*Exercice*. Comment adapter l'algorithme précédent pour calculer directement l'ensemble dominant de taille optimale ?]

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 $(3 - \varepsilon)^\omega \cdot n^{O(1)}$ pour une certaine constante $\varepsilon > 0$, alors le problème Sat peut être résolu en temps $(2 - \delta)^n \cdot n^{O(1)}$ pour une certaine constante $\delta > 0$, n étant le nombre de variables de la formule. Comme personne ne croit à cette dernière affirmation, tout le monde pense que « 3 » est la meilleure constante possible pour la base de l'exponentielle en ω pour Ensemble dominant pour les graphes de largeur arborescente ω.

En fait toute propriété de graphe exprimable par une formule Φ de logique mona-

dique du second ordre [45] (MSO_1) ou de son extension [46] MSO_2, est FPT en la largeur arborescente du graphe. Les problèmes comme HAMILTONISME, 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 ont peut lire aussi la preuve dans [FG06]. C'est devenu l'archétype des méta-théorèmes algorithmiques.

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 fixées. Malheureusement, cette constante dépend de manière super-exponentielle en le nombre $h(\Phi)$ d'alternances de quantificateurs $\forall...\exists...$ de la formule Φ :

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

En pratique un algorithme construit « à la main » sera bien plus efficace que la méthode générale taillée pour une formule Φ quelconque. Plusieurs extensions du théorème de Courcelle ont été proposé, allant de l'extension de la classe des graphes (ceux de *clique-width* bornée) et aussi de la logique (logique avec compteurs comme $CMSO_1$ et $CMSO_2$). Voir [CE12] ou Wikipédia pour plus de détails.

2.10.5 Application aux graphes planaires

Les deux prochaines sections sont consacrées au problème ENSEMBLE DOMINANT de taille k dans les graphes planaires. 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 2.7.3 et le théorème 2.4), basé sur un algorithme récursif et la réduction de données. Le terme exponentielle en k était $f(k) = 8^k$, grâce à une propriété non-triviale. L'objectif va être de produire un algorithme FPT avec $f(k) = 2^{o(k)}$ ce qui est théoriquement bien plus petit que c^k pour toute constante $c > 1$.

On va ici construire un algorithme très différent, qui sera aussi de complexité linéaire en n au lieu de quadratique comme dans le théorème 2.4. Pour cela, on va se servir des décompositions arborescentes des graphes planaires et relier la largeur arborescente et la taille des ensembles dominants. Malheureusement, comme vu page 86, les graphes planaires ne sont pas, en général, de largeur arborescente bornée. Par exemple, une grille $p \times p$ est de largeur arborescente p.

45. Il s'agit de formules avec des quantificateurs du second ordre ($\forall$ et $\exists$) pouvant porter sur des ensembles de sommets (prédicats unaires) mais pas d'arêtes (prédicats binaires). Typiquement, la 3-COLORATION d'un graphe G s'exprime par la formule $\Phi(G) = \exists X,Y,Z,\forall u \in V(G),(u \in X \vee u \in Y \vee u \in Z) \wedge \forall uv \in E(G),\neg((u \in X \wedge v \in X) \vee (u \in Y \wedge v \in Y) \vee (u \in Z \wedge v \in Z))$.

46. Dans son extension, les quantificateurs du second ordre peuvent s'appliquer aux ensembles d'arêtes, mais pas plus. Cela permet d'exprimer la notion de cycle de taille non borné par exemple.

Théorème 2.12 *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 le facteur exponentiel est ici $f(k) = 3^{9k} = \left(3^9\right)^k = 19683^k$. Certes c'est moins intéressant que 8^k mais est premier pas vers un algorithme en $2^{o(k)}$ basé sur la même méthodologie.

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 2.10.3 et 2.10.4. Toute la suite de la section est consacrée à la preuve du théorème 2.12.

Lemme 2.4 *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 $\leqslant 3h$.*

En particulier, si G est planaire et de diamètre D, alors $\mathrm{tw}(G) \leqslant 3D$.

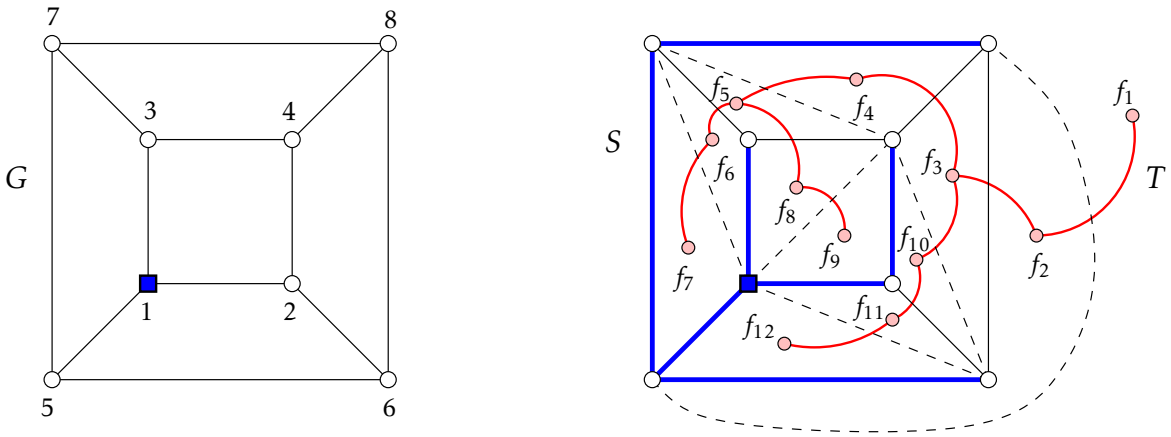
Preuve. Soit G un graphe planaire dessiné dans le plan et soit S un arbre couvrant de hauteur h. On commence par trianguler le graphe G en ajoutant autant que possible des arêtes aux faces de G toute en préservant la planarité. Il est possible de le faire en temps $O(n)$, G ayant au plus $O(n)$ faces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

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, $|X_f| \leqslant 3h+1$. L'arbre T de la décomposition est défini par : (1) la collection de sacs $\{X_f : f \text{ face de } G\}$; (2) les 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 $C = X_{f_1}, \ldots, X_{f_k}$ 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 2.44). 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.



$X_{f_1} = \{1,5,7,8\}$
$X_{f_2} = \{1,5,6,7,8\}$
$X_{f_3} = \{1,2,4,5,6,7,8\}$
$X_{f_4} = \{1,2,4,5,7,8\}$
$X_{f_5} = \{1,3,2,4,5,7\}$
$X_{f_6} = \{1,3,5,7\}$
$X_{f_7} = \{1,5,7\}$
$X_{f_8} = \{1,2,3,4\}$
$X_{f_9} = \{1,2,4\}$
$X_{f_{10}} = \{1,2,4,5,6\}$
$X_{f_{11}} = \{1,2,5,6\}$
$X_{f_{12}} = \{1,5,6\}$

FIGURE 2.43 – Construction d'une décomposition arborescente T (en rouge) de largeur $\leqslant 3h$ pour un graphe planaire G à partir d'un arbre couvrant S (en bleu) de profondeur $h = 3$. En fait, $\mathrm{tw}(G) = 3$ car il contient K_4 comme mineur. [*Question*. Donnez un modèle de K_4.]

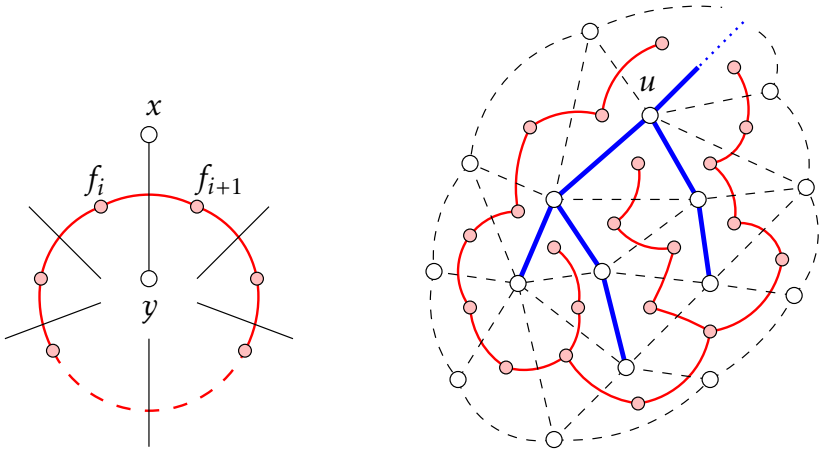


FIGURE 2.44 – T ne contient pas de cycle, et les sacs contenant u induisent un sous-arbre de T.

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 (toute arête est le bord d'une face). 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 2.44.

Par exemple, sur l'exemple de la figure 2.43, le sommet 7 a pour descendant 8 dans S. Les faces incidentes à 7 ou 8 sont $f_1, f_2, f_3, f, f_5, f_6, f_7$ qui forment en fait un chemin dans T. □

Lemme 2.5 *Soit G un graphe possédant un plus court chemin de longueur h et un ensemble dominant de taille k. Alors $h \leqslant 3k-1$.*

Preuve. Soit P un plus court chemin de longueur h 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.

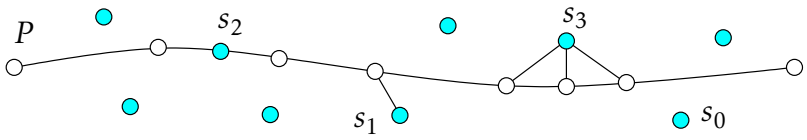


FIGURE 2.45 – Chaque $s_i \in S$ domine au mieux trois sommets d'un plus court chemin P.

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 2.45). Donc si $|V(P)| > 3|S|$, alors un sommet de P ne serait pas dominé. Il suit que $h \leqslant 3k-1$, car $|V(P)| = h+1 \leqslant 3|S| = 3k$. □

[*Exercice*. Généralisez le lemme 2.5 au cas d'un chemin P de longueur h, qui n'est pas forcément un plus court chemin, mais qui a la propriété que pour toute paire de sommets x, y de P, si x, y sont voisins dans G^2, alors x et y sont à distance au plus t dans P. On rappelle, cf. page 113, que G^2 est le graphe G dans lequel on ajoute tous les arêtes entre sommets à distance deux.]

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

Algorithme Planar-DomSet_v1(G,k)

1. Calculer en temps $O(n)$ la hauteur h d'un arbre en largeur d'abord.
2. Si $h > 3k-1$, renvoyer FAUX (lemme 2.5).
3. Calculer une décomposition arborescente T de largeur $\omega \leqslant h$ (lemme 2.4).
4. Résoudre, grâce à T, ENSEMBLE DOMINANT de taille k en temps $O(3^\omega \omega^2 \cdot n)$ (théorème 2.11).

L'étape 2 se justifie par le lemme 2.5 et le fait que dans un arbre en largeur d'abord, les branches partant de la racine sont des plus cours chemins. Donc si G possède un ensemble dominant de taille k, alors la hauteur h de tout BFS vérifie $h \leqslant 3k-1$. On note qu'après l'étape 2, la largeur $\omega \leqslant 3h \leqslant 3(3k-1) = 9k-3$. Donc la complexité totale de l'algorithme est $O(n+nh+3^{9k}k^2 \cdot n) = O(3^{9k}k^2 \cdot n)$, car $n+nh = O(nk) = O(3^{9k}k^2 \cdot n)$, ce qui termine la preuve du théorème 2.12.

[*Exercice*. On se propose d'étudier la variante suivante de l'algorithme Planar-DomSet_v1 : À l'étape 1, on calcule plutôt un arbre DFS de hauteur h. À l'étape 3, on produit une décomposition arborescente de largeur h grâce à l'heuristique vue page 90, mais aussi la décomposition arborescente de largeur $3h'$ issue d'un arbre BFS comme dans l'algorithme d'origine. On choisit alors la meilleure des deux, soit une décomposition arborescente de largeur $\omega \leqslant \min\{h, 3h'\}$. Q1. Quelle est alors la complexité de cette variante ? Q2. Quel est l'inconvénient de cette variante ?]

2.10.6 Amélioration

On peut encore réduire la complexité de l'algorithme précédent pour Ensemble dominant de taille k pour les graphes planaires, en diminuant le terme $2^{O(k)}$. On va le remplacer par $2^{O(\sqrt{k})}$.

L'idée générale est somme toute assez simple. On essaye de construire une décomposition arborescente de largeur t en temps $2^{O(t)} \cdot n^{O(1)}$ pour un seuil t qui est une fonction du paramètre k, typiquement $t \approx \sqrt{k}$. Si la construction a échouée, c'est-à-dire si la largeur arborescente est trop grande par rapport au seuil, on renvoie Faux. Sinon, on résout le problème avec la programmation dynamique sur la décomposition arborescente ainsi calculée en temps $2^{O(t)} \cdot n^{O(1)}$.

La principale difficulté est donc de montrer que si $\mathrm{tw}(G)$ est plus grande qu'un certain seuil, alors on peut effectivement conclure qu'il n'a pas d'ensemble dominant de taille k. Pour cela on utilise les deux lemmes suivants. Le second (lemme 2.7), le plus simple, utilise le premier (lemme 2.6) qui lui est très profond. Sa preuve complète sort largement du cadre de ce cours. On admettra donc ce résultat qui permet de justifier l'analyse de l'algorithme, mais qui ne change pas le principe de l'algorithme.

Lemme 2.6 ([RST94, p. 346] et [GT12]) *Soit G un graphe planaire. Si* $\mathrm{tw}(G) > (9s-11)/2$, *alors G contient une grille $s \times s$ comme mineur.*

La contraposée du lemme 2.6 affirme donc qu'un graphe planaire qui n'a pas de grille $s \times s$ comme mineur a une largeur arborescente au plus $(9s-11)/2 = O(s)$.

Notons que le contraire du lemme 2.6 est bien plus simple à démontrer à savoir : si G possède une grille $s \times s$ comme mineur, alors $\mathrm{tw}(G) = \Omega(s)$. En effet, d'une part on peut montrer que la largeur arborescente d'une grille $s \times s$ est s. Et d'autre part, $\mathrm{tw}(G) \geqslant \mathrm{tw}(H)$

pour tout mineur H de G puisque la famille des graphes de largeur arborescence $\leqslant k$ est close par mineur (cf. aussi la note de bas de page [28]).

Une version plus faible du lemme 2.6, avec la condition $\mathrm{tw}(G) = \Omega(s^2)$ force l'apparition d'une grille $s \times s$ comme mineur, avait été présentée dix ans plus tôt dans [RS84]. Il est conjecturé dans [GUM10] que le terme « $(9s-11)/2$ » du lemme 2.6 peut être remplacé par « $2s + o(s)$ ». Il est démontré dans [GT12] que ce terme est au moins $2(s-1)$, à cause du cylindre $2s \times s$, le produit d'un cycle de taille $2s$ et d'un chemin de taille s. Ce cylindre est de largeur arborescente $2s$ et contient une grille $s \times s$. 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

$$\mathrm{tw}(G) \leqslant 20^{4(|V(H)|+2|E(H)|)^5}. \tag{2.2}$$

On en déduit par exemple que les graphes sans K_4 comme mineur sont de largeur arborescente au plus $2^{4(4+2\cdot 6)^5} \approx 2^{400\,000}$. En fait ces graphes sont exactement les sous-graphes des graphes série-parallèles et ont une largeur arborescente au plus 2. L'équation (2.2) a été récemment améliorée en $\mathrm{tw}(G) \leqslant |V(H)|^{O(1)}$ par [CT19]. En fait, il est aussi montré que si $\mathrm{tw}(G) > s^{9+o(1)}$ alors G contient une grille $O(s) \times O(s)$ comme mineur. Notez bien qu'ici G est un graphe quelconque. Il est d'ailleurs conjecturé que l'exposant « 9 » peut être remplacé par « 2 » et que c'est le meilleur exposant possible.

Lemme 2.7 *Soit G un graphe planaire possédant une grille $\ell \times \ell$ comme mineur et un ensemble dominant de taille k. Alors $\ell \leqslant 3\sqrt{k} + 2$.*

Preuve. On considère un modèle de cette grille $\ell \times \ell$ 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 2.46. 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 et qui ont comme plus proche ancêtre un sommet de X. Voir la figure 2.47.

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 super-nœuds puisque chaque super-nœuds contient au moins un sommet de G.

Le point clef, liée à la planarité de G, est que tout sommet de S domine au plus 9 super-nœuds internes de la grille, ceux qui ne sont pas sur le bord de la grille. Pour le montrer, on va supposer que $\ell > 2$, puisque sinon la question ne se pose pas, le modèle n'a pas de super-nœud interne. Notons que si $\ell \leqslant 2$, alors $\ell \leqslant 3\sqrt{k} + 2$ quel que soit k.

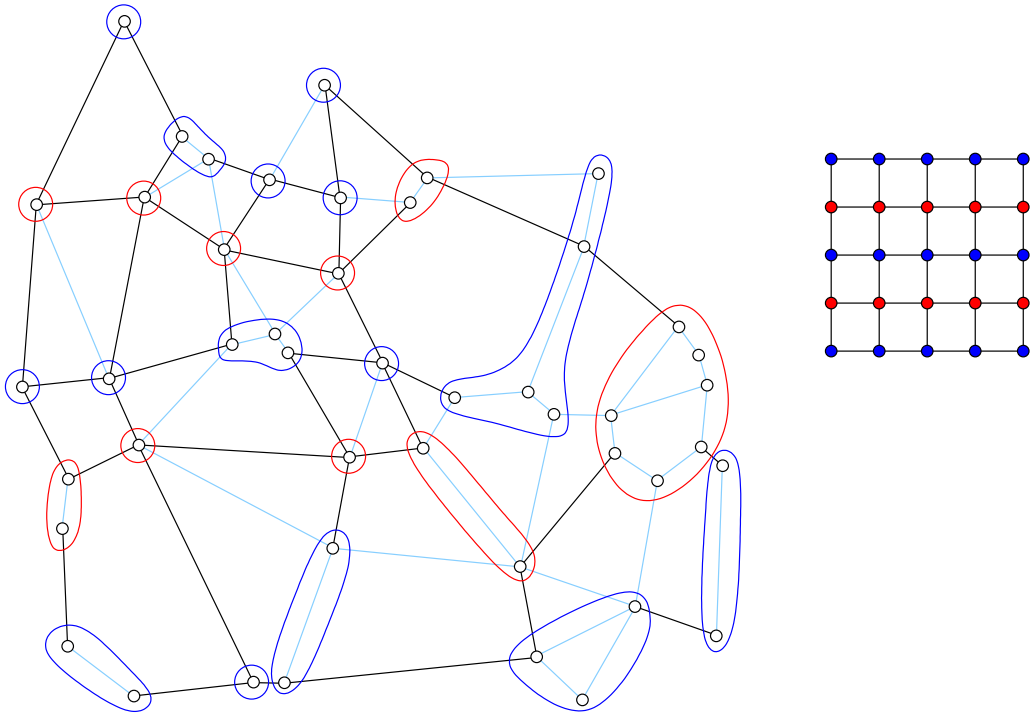


FIGURE 2.46 – Une grille 5×5 mineur d'un graphe de Gabriel à 45 sommets (les arêtes claires sont à supprimer ou à contracter). Les sommets sont des points du plan, et deux points u, v sont adjacents ssi l'intérieur du disque de diamètre $|uv|$ ne contient aucun autre point.

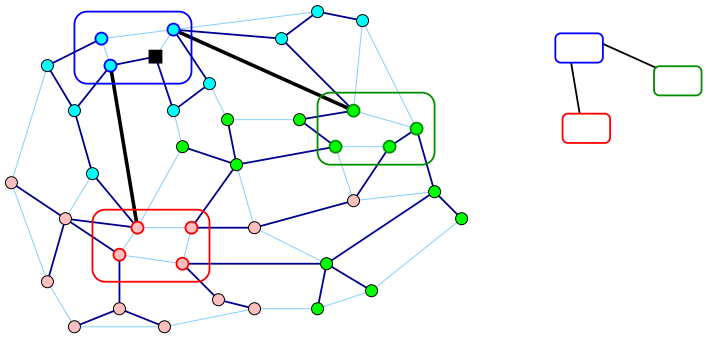


FIGURE 2.47 – Construction d'une partition des sommets du graphe à partir des super-nœuds du mineur (représenté à droite) et d'un arbre couvrant.

Soit $s \in S$. Si $s \notin V(G')$, alors la grille et s ne sont pas dans la même composante connexe et donc s ne domine aucun super-nœud. Sinon, $s \in V(G')$ et il appartient donc à un super-nœud (x, y).

Remarquons qu'à tout cycle C de la grille on peut construire un cycle de G' qui ne traverse que les super-nœuds et les arêtes correspondantes aux sommets et arêtes de la grille traversés par C. (Rappelons que les super-nœuds sont connexes.)

Deux cas de figure se présentent (cf. figure 2.48) :

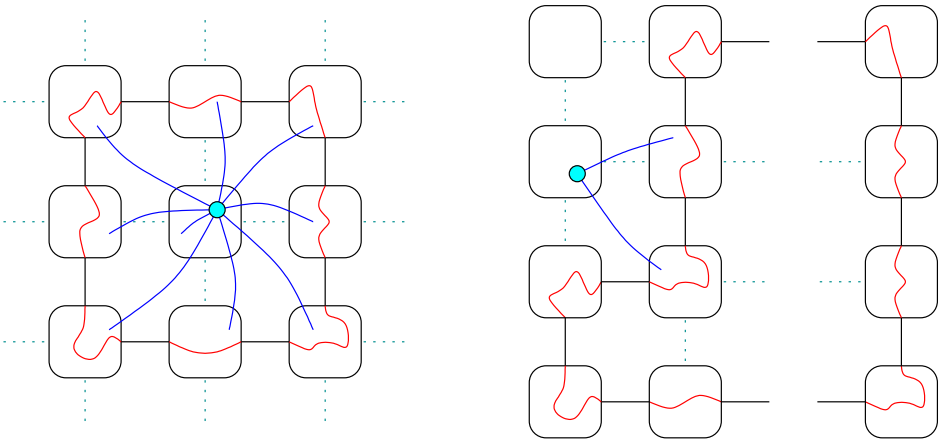


FIGURE 2.48 – Super-nœuds internes dominés par un sommet $s \in S$. Les arêtes incidents à s ne peuvent traverser le cycle C. Attention ! s pourrait dominer un nombre non bornée de super-nœuds du bord de la grille.

Le super-nœud (x,y) est 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 région où les voisins de s doivent se situer (courbe de Jordan). Cette région comprend au plus 9 super-nœuds internes.

Le super-nœud (x,y) est sur le 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 $\ell \geqslant 3$. 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 2.48).

On a donc montré que dans tous les cas s domine au plus 9 super-nœuds internes. Le mineur possède $(\ell-2)^2$ super-nœuds internes. Donc $(\ell-2)^2 \leqslant 9|S| = 9k$ puisque sinon un super-nœud (et donc certains sommets de G) ne seraient pas dominés par S. Il suit que $\ell \leqslant 3\sqrt{k}+2$. □

Des deux lemmes 2.6 et 2.7, on déduit que si G est planaire et possède un ensemble dominant de taille k, alors $\mathrm{tw}(G) < \frac{27}{2}\sqrt{k}+8$. En effet, soit ℓ la taille de la plus grande grille carrée mineure de G. Dit autrement, G ne contient pas de grille $(\ell+1)\times(\ell+1)$ comme mineur. D'après le lemme 2.6 (sa contraposée avec $s = \ell+1$) :

$$\mathrm{tw}(G) \leqslant \frac{9s-11}{2} = \frac{9(\ell+1)-11}{2} = \frac{9\ell}{2}-1 \leqslant \frac{9}{2}\left(3\sqrt{k}+2\right)-1 = \frac{27}{2}\sqrt{k}+8\,.$$

On en déduit donc l'algorithme suivant :

Algorithme Planar-DomSet_v2(G,k)

1. Soit $t := \frac{27}{2}\sqrt{k}+8$.
2. Déterminer en temps $2^{O(t)}\cdot n$ si $\mathrm{tw}(G) > t$ ou sinon construire une décomposition arborescente T de largeur $\omega \leqslant 2t+1$ (théorème 2.9).
3. Si $\mathrm{tw}(G) > t$, alors renvoyer Faux (lemme 2.7).
4. Résoudre Ensemble dominant grâce à T en temps $O(3^\omega \omega^2 \cdot n)$ (théorème 2.11).

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

Théorème 2.13 *Ensemble dominant de taille k pour les graphes planaires à n sommets peut être résolu (et un tel ensemble peut être construit) en temps $2^{O(\sqrt{k})}\cdot n$.*

Il faut remarquer qu'en utilisant l'algorithme donnant une approximation à un facteur 2 sur la largeur arborescente (+1), 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 :

$$3^\omega \leqslant 3^{2t+1} \approx 3^{27\sqrt{k}} = \underbrace{7\,625\,597\,484\,987}_{\text{13 chiffres}}{}^{\sqrt{k}}.$$

L'algorithme fpt pour Ensemble dominant dans les graphes planaires ayant la plus faible fonction $f(k)$, tout en étant au plus cubique, a une complexité en temps de $O(35\,860^{\sqrt{k}}k + n^3)$ [FT06]. Ce résultat est obtenu en combinant les techniques précédentes avec celle du noyau vu au paragraphe 2.8. 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]. Même pour ce dernier algorithme, on a $f(k) > 10^{10}$ dès que $k = 8$.

2.11 Remarques finales

L'algorithme en $O(35860^{\sqrt{k}}k + n^3)$ pour Ensemble dominant de taille k dans les graphes planaires doit être comparé à celui en $O(8^k n^2)$ (cf. le théorème 2.4), qui peut en fait être amélioré en $O(8^k n)$ (voir [AFF+05]). Ce dernier est plus rapide lorsque k est « assez petit », et surtout plus simple à programmer [47]. Le goulot d'étranglement pour la technique des décompositions arborescentes reste le calcul de la décomposition.

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

On remarque aussi qu'une complexité en $c^\omega \cdot \text{poly}(n)$, pour une certaine constante $c > 1$, obtenue par programmation dynamique sur une décomposition arborescente de largeur ω, devient polynomiale dès que $\omega = O(\log n)$ et pas seulement lorsque $\omega = O(1)$. Ainsi des travaux récents, comme [ACHS21], essayent d'identifier de grandes familles de graphes ayant une largeur arborescente logarithmique en la taille du graphe (et si possible de construire de telles décompositions). Par exemple, tout graphe sans K_3 ni subdivision de $K_{2,3}$ comme sous-graphe induit est de largeur arborescente logarithmique.

Enfin, signalons que parfois des algorithmes FPT sont utilisés à la place d'algorithmes polynomiaux. Par exemple, un algorithme en $O(2^k n)$ peut être bien plus rapide, lorsque k est assez petit devant n, qu'un algorithme en $O(n^6)$ ou cubique en le nombre d'arêtes. Ici « assez petit » voudra dire $k < 5 \log n$. Pour $n = 1\,000$ par exemple, l'algorithme exponentiel sera plus rapide pour $k \leqslant 50$, valeur déjà intéressante. Notez aussi qu'exécuter 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 tel exemple est le calcul classique du flot 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 $O(nmC)$ où C est la valeur du flot maximum. Potentiellement C peut être exponentiel en n et même non borné. Cet algorithme est FPT en C. 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 C donc. Lorsque $C = o(m)$, ce qui est déjà beaucoup en pratique, Ford-Fulkerson sera toujours plus rapide qu'Edmonds-Karp. L'état de l'art sur les algorithmes de flot maximum est donné page 133.

2.12 Algorithme progressif pour Ensemble dominant

On va présenter un algorithme qui donne des résultats intéressant pour Ensemble dominant pour des familles de graphes plus large que celles déjà rencontrées, en particulier la famille des graphes qui excluent un minor fixé. Ce type d'algorithmes, proposés récemment dans [FPST18], sont dits *progressifs* car ils construisent la solution progressivement, par essais-erreurs successifs, en tirant partie des témoins ayant produit l'erreur.

Ces algorithmes sont relativement simples contrairement à leur analyse. En gros ils marchent bien mais c'est difficile de voir pourquoi. En fait, avec le même principe, on peut résoudre des généralisations d'Ensemble dominant et même d'Ensemble indépendant, à savoir Ensemble dominant à distance r et Ensemble indépendant à distance r. Pour la première variante, il s'agit de trouver un ensemble S tel que tout sommet est à distance au plus r de S ; et pour la deuxième, un ensemble S tel qu'il n'y a pas deux sommets à distance r ou moins. Les versions classiques correspondent à $r = 1$. Rappelons qu'on ne sait pas si ces problèmes sont FPT en k (la taille de l'ensemble) ; on ne sait

pas faire mieux que $n^{\Omega(k)}$; et ils restent NP-complets même pour les graphes planaires.

2.12.1 L'algorithme

On ne présente l'algorithme seulement pour Ensemble dominant à distance r. Étant donnés deux sous-ensembles de sommets S et W d'un graphe G, on dira que S *domine* W *à distance* r si tous les sommets de W sont à distance au plus r d'un des sommets de S dans G.

L'algorithme essaye de construire en parallèle une solution S et un témoin [48] W de la non-existence de S comme ensemble dominant à distance r. La figure 2.49 illustre l'exécution de l'algorithme.

Algorithme Progress(G, r, k)

1. $W := \varnothing$
2. Répéter :
   (a) Chercher un ensemble S de taille k qui domine W à distance r.
   (b) Si S n'existe pas, renvoyer Faux.
   (c) Si S domine $V(G)$ à distance r, alors renvoyer Vrai.
   (d) Ajouter à W un sommet qui n'est pas dominé à distance r par S.

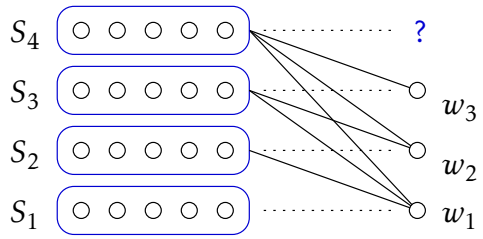


Figure 2.49 – Exécution schématique de l'algorithme Progress pour $k = 5$. Les ensembles S_i et les sommets w_i sont ceux sélectionnés au cours de la i-ème itération. Bien sûr les problèmes arrivent lorsque $|W| > k$ puisque sinon $S = W$ est une solution valide.

Il assez facile de ce convaincre que l'algorithme est correct. En effet, pour dominer $V(G)$ il faut déjà dominer W. La seule question porte sur les performances de l'algorithme, en particulier sur le nombre d'itérations et la complexité de l'étape 2a. Il n'y a aucune raison pour que l'algorithme termine avec $|W| \leqslant f(k)$ pour une certaine fonction f. On pourrait très bien avoir $|W| = \Omega(n)$. Et bien sûr, plus $|W|$ est grand, plus l'étape 2a va devenir couteuse.

Cependant, l'intuition est que si le graphe est plutôt « plat » alors les sommets de W vont avoir tendances à être choisis loin les uns des autres, et donc on pourrait vite

48. Comme *witness* en Anglais.

arriver à sortir de la boucle (avec une réponse positive ou négative, selon le cas).

Concernant la complexité de l'étape 2a, nous avons :

Proposition 2.3 *L'étape 2a de Progress peut être exécutée en temps* $O(|W| \cdot (n+m) + |W|2^{k|W|})$.

Cela montre au passage que le sous-problème résolu à l'étape 2a, qui est en fait une généralisation du problème initial où $W = V(G)$, est FPT en k et $|W|$. Et pour $r = 1$, c'est aussi exactement le problème ENSEMBLE DOMINANT ANNOTÉ rencontré page 54, où les sommets non marqués (à dominer donc) sont ceux de W.

Comme les autres étapes peuvent être exécutées en temps $O(n+m)$, il suit que la complexité de l'algorithme, après un total de t itérations, est de $O(t^2(n+m) + t2^{kt})$ puisque à la i-ème itération $|W| = i$ et que $\sum_{i=0}^{t-1} i2^{ki} = O(t2^{kt})$.

Preuve. Soit $W = \{w_1, w_2, w_3, \ldots\}$. À chaque sommet u de G on associe son *profile* pour W, un mot binaire noté $\mathsf{profile}(u)$ indiquant, pour chaque indice i, si u domine w_i à distance r. Plus formellement,

$$\forall u \in V(G), \quad u \mapsto \mathsf{profile}(u) \in \{0,1\}^{|W|}$$

$$\forall i \in \{1,\ldots,|W|\}, \quad \mathsf{profile}(u)[i] := \begin{cases} 1 & u \text{ domine } w_i \text{ à distance } r \\ 0 & \text{sinon} \end{cases}$$

Le profile de chaque sommet peut être calculé en exécutant un parcours BFS depuis chaque $w_i \in W$. Lors des parcours, tous les sommets d'un niveau $\leqslant r$ reçoivent 1 dans le bit i de leur profile, les autres reçoivent 0. Voir la figure 2.50 pour un exemple. Cela prend un temps $O(|W| \cdot (n+m))$. Notons qu'en pratique on peut tronquer le parcours BFS à la profondeur r en mettant un 0 à tous les profiles de sommets non parcourus.

La remarque importante est que pour chercher l'ensemble S dominant W à distance r il est inutile de considérer les sommets ayant le même profile, un seul par profile suffit. En effet, les sommets de W dominés par chaque $u \in S$ sont déterminés par le profile de u. Et donc si $u, v \in S$ et si $\mathsf{profile}(u) = \mathsf{profile}(v)$, alors S et $S \setminus \{v\}$ domineront (à distance r) exactement les mêmes sommets de W. Il suffit donc de chercher S dans un ensemble $R \subseteq V(G)$ de représentants de chacun des profiles.

Le calcul de R peut se faire ainsi : (1) on trie les sommets selon leur profile, c'est-à-dire on trie les clefs $(\mathsf{profile}(u), u)$; et (2) on balaye cette liste pour ne garder qu'un seul sommet par profile, le premier rencontré.

L'étape de tri effectue $O(n \log n)$ comparaisons chacune consistant à comparer des mots de $|W|$ bits. Dans le modèle RAM, les registres sont de $b = \Omega(\log n)$ bits. Donc en découpant les mots en blocs de b bits, chaque comparaison prend un temps $O(|W|/b) = O(|W|/\log n)$ correspondant au nombre de blocs dans un mot. Au total le tri prend un temps de $O(n \log n \cdot |W|/\log n) = O(n|W|)$. L'étape de balayage pour construire R prend

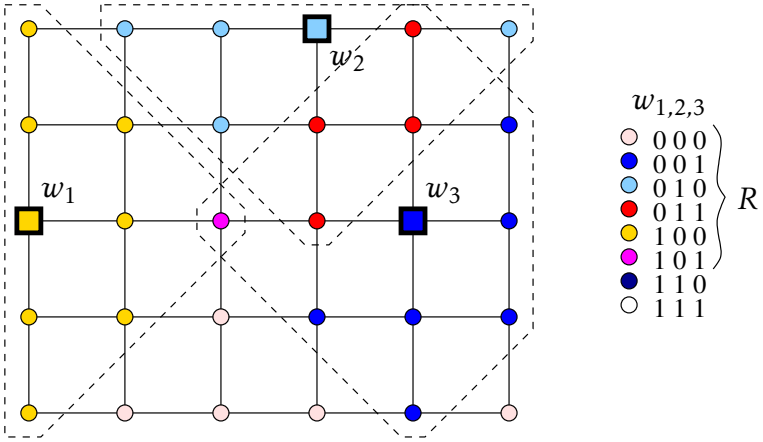


FIGURE 2.50 – Profiles (=couleurs) des sommets pour $r = 2$ et $W = \{w_1, w_2, w_3\}$ (sommets carrés). Il n'y a que six profiles différents (ensemble R), qui peut même se réduire à seulement deux qui ne sont pas inclus l'un dans l'autre. [*Question.* Lesquels ?] Un ensemble S de taille deux est nécessaire et suffisant pour dominer W à distance r.

elle un temps $O(n \cdot |W|/b)$ car il suffit de faire des comparaisons entre clefs qui se suivent dans la liste triée.

On peut maintenant chercher S dans R. Malheureusement, on ne sait pas faire bien mieux que la méthode *brute-force*. Comme déjà vue dans la section 2.3, cela coûte $\binom{|R|}{|S|} = \binom{|R|}{k} < (e|R|/k)^k = k(e/k)^k \cdot |R|^k/k = O(|R|^k/k)$ tests [49]. Notons que le test de validité d'un ensemble S revient à vérifier que l'union des sommets de W qui sont dominés par ceux de S vaut W. En terme de mot binaire cela revient à vérifier :

$$\bigvee_{u \in S} \mathsf{profile}(u) \;=\; \underbrace{111\ldots11}_{|W|} \,.$$

Cela prend un temps $O(|S| \cdot |W|/b) = O(k|W|)$ par test, soit $O(k|W|) \cdot O(|R|^k/k) = O(|W| \cdot |R|^k)$ pour la recherche de S.

Notons qu'en pratique, avant de lancer cette recherche exhaustive coûteuse de S dans R, on a intérêt de « nettoyer » R en enlevant les sommets de profile inutile. Comme par exemple le profile nul qui ne peut pas être dans la solution optimal S. Plus généralement on peut supprimer les profiles qui sont inclus dans d'autres. En terme de mot binaire, l'inclusion [50] se teste facilement : $X \subseteq Y \Leftrightarrow$ `(X&Y)==X` $\Leftrightarrow$ `(X|Y)==Y`. Cela prend un temps $O(|W| \cdot |R|^2)$. [*Question.* Comment ?] On appliquera ce nettoyage que si

49. On peut montrer que la fonction $x \mapsto x \cdot (e/x)^x$ atteint un maximum de ≈ 3.78 pour tout $x > 0$.

50. Le codage des sous-ensembles par des mots binaires a déjà été abordé en Licence dans l'implémentation par programmation dynamique de l'algorithme de Held-Karp pour le TSP [Gav23, Chapitre 2, section 3.3].

$O(|W|\cdot|R|^2) = O(|W|\cdot|R|^k)$, c'est-à-dire si $k \geqslant 2$, de façon à absorber ce temps par celui de la recherche de S dans R. [*Exercice*. Donnez, dans l'exemple de la figure 2.50, le plus petit sous-ensemble de R de profiles deux-à-deux non-inclus.]

La remarque finale est que $|R| \leqslant 2^{|W|}$ puisqu'il s'agit de mots de $|W|$ bits. En fait, c'est un peu moins car le nombre maximum de mots binaires non deux-à-deux inclus est $\max_i \binom{|W|}{i} = \binom{|W|}{|W|/2} = 2^{|W|}/\Theta(\sqrt{|W|})$. Au total, la complexité pour la cherche de S est donc (calcul des BFS, calcul de R, recherche de S) :

$$O\Big(|W|\cdot(n+m)+n|W|+|W|\cdot|R|^k\Big) = O\Big(|W|\cdot(n+m)+|W|\cdot|R|^k\Big) \tag{2.3}$$

$$= O\Big(|W|\cdot(n+m)+|W|\cdot 2^{k|W|}\Big)\,. \tag{2.4}$$

□

En fait l'étape 2a s'exécute encore plus rapidement sur certaines familles de graphes, notamment parce que le nombre de profiles différents peut être beaucoup plus faible que $2^{|W|}/\Theta(\sqrt{|W|})$.

[*Exercice*. Que vaut, en fonction de $|W|$, le nombre de profiles si G est un chemin ?]

[*Exercice*. On considère la variante ProgressOpt(G,r) où l'étape 2a de l'algorithme Progress est remplacée par le calcul d'un ensemble S de taille minimum dominant W à distance r (l'étape 2b étant devenue obsolète). Montrez que cette variante calcule un ensemble dominant à distance r de taille minimum pour G. Et que se passe-t-il si dans la même étape on approxime la taille optimale de S à un facteur α près ?]

En fait le nombre de profiles est lié à la dimension de Vapnik and Chervonenkis d'un certain hyper-graphe défini à partir de G, celui des boules d'un rayon donnée dans G où les hyper-arêtes sont simplement toutes ces boules. On parle de *distance VC-dimension* de G, notion largement abordée dans [DHV19]. Si cette dimension est d, alors par le lemme de Sauer-Shelah, il ne peut avoir que $\sum_{i=0}^{d}\binom{|W|}{i} = O(|W|^d)$ profiles différents. Quel que soit le graphe, la dimension est toujours au plus $\log_2 n$, donnant le nombre de profiles en $2^{|W|}$ dans le pire des cas comme considérée dans la proposition 2.3. Il est connu que les graphes excluant H comme mineur sont de *distance VC-dimension* au plus $|V(H)|-1$, soit au plus 4 pour les graphes planaires. [*Question*. Pourquoi 4 ? Que vaut cette dimension pour un arbre ?]

De manière générale, la *VC-dimension* d'un hyper-graphe $(B,\mathcal{F})$, où B représente les sommets et $\mathcal{F}$ une famille de sous-ensembles de B (les hyper-arêtes), est définie comme la taille du plus grand sous-ensemble $X \subset B$ *pulvérisé* par $\mathcal{F}$, c'est-à-dire si

$$\forall Y \subseteq X,\quad \exists Z \in \mathcal{F},\quad X \cap Z \;=\; Y\,.$$

Dit autrement, les $2^{|X|}$ sous-ensembles possibles de X, c'est-à-dire les profiles, peuvent apparaître en considérant les intersections avec des ensembles pris dans $\mathcal{F}$. [*Exercice*. Quelle est la VC-dimension de l'hyper-graphe $(B,\mathcal{F})$ donné par la figure 3.13(a).] [*Exercice*. Pourquoi de manière générale la VC-dimension de $(B,\mathcal{F})$ est toujours $\leqslant \log|\mathcal{F}|$?]

2.12.2 Graphes nul part denses

Avant de définir la famille de graphes pour laquelle l'algorithme est performant, introduisons deux variations autour des mineurs de graphes, voir les figures 2.51.

- Un graphe H est un *mineur topologique* d'un graphe G si on peut obtenir H à partir d'un sous-graphe de G et en contractant des arêtes incidentes à un sommet de degré deux.
- Un graphe H est un *mineur de profondeur* ρ d'un graphe G si on peut obtenir H à partir d'un sous-graphe de G et en contractant des sous-graphes disjoints de rayon [51] au plus ρ.

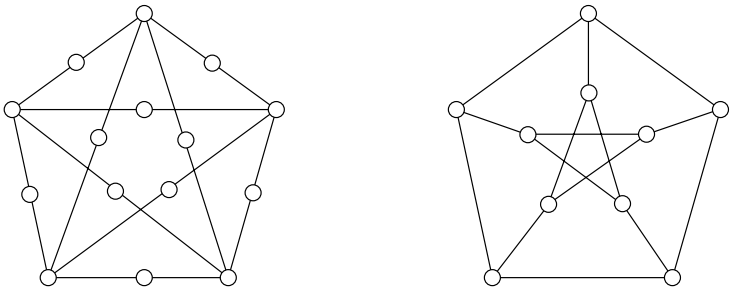


FIGURE 2.51 – À gauche, une subdivision de K_5 contenant K_5 comme mineur de profondeur 1 et aussi comme mineur topologique. À droite le graphe de Petersen, contient aussi K_5 comme mineur de profondeur 1, mais pas comme mineur topologique. [*Exercice*. Construire les modèles évoqués. Pourquoi K_5 n'est pas un mineur topologique du graphe de Petersen ?]

Les mineurs topologiques et les mineurs de profondeur ρ sont donc des cas particuliers de mineurs. Une définition équivalente pour un mineur topologique est de dire qu'une subdivision [52] de H est un sous-graphe de G. Les notions de mineurs et de mineurs topologiques correspondent lorsque le mineur H est de degré au plus trois. [*Exercice*. Pourquoi ?] Mais si H est un mineur topologique de G, alors $\deg(H) \leqslant \deg(G)$, ce qui n'est pas forcément vrai si H est simplement un mineur, comme le montre la figure 2.25 ou encore une grille $(k+2)\times(k+2)$ contenant une étoile de degré $4k$ comme mineur.

Les mineurs de faible profondeur sont les structures que l'on peut obtenir à partir du graphe initial par modifications locales. En effet, suppression de sommets, suppression d'arêtes et contraction de sous-graphes de rayon borné sont des modifications locales.

Remarquons qu'un mineur de profondeur nulle n'est rien d'autre qu'un sous-graphe de G. Et qu'un mineur n'est rien d'autre qu'un mineur de profondeur infinie. Le para-

51. Rappel : Un graphe est de rayon ρ s'il possède un arbre couvrant de profondeur ρ. Notez qu'il est forcément connexe.

52. Une subdivision d'un graphe H est un graphe S obtenu en subdivisant chaque arêtes de H, en une ou plusieurs arêtes.

mètre ρ permet donc de faire varier graduellement la notion de sous-graphe à la notion de mineur. Les graphes excluant un graphe fixé H comme mineur de profondeur bornée sont aussi appellés *shallow-minor free graphs*. Alors que ceux excluant H comme mineur sont appellés *minor-free graphs*.

Définition 2.7 (Dense nul part) *Une famille de graphe $\mathcal{F}$ est* dense nul part *(en Anglais nowhere dense) s'il existe deux constantes $\rho, \delta \in \mathbb{N}$ telles tout graphe de $\mathcal{F}$ exclut K_δ comme mineur de profondeur ρ.*

En quelque sorte, ces graphes ne contiennent pas de structures localement denses (comme des cliques suffisamment grosses), d'où le nom. Il existe de très nombreuses caractérisations et définitions équivalentes pour ces familles. On peut citer [NOdM08a][NOdM08b][NOdMW12] par exemple.

Supposons qu'un graphe G exclut un graphe H comme mineur de profondeur ρ, alors :

- G exclut H comme mineur de profondeur $> \rho$, en particulier G exclut H comme mineur. [*Question*. Pourquoi ? Est-ce qu'il exclut H comme mineur topologique ?]
- G exclut tout *sur-graphe*[53] de H comme mineur de profondeur ρ, en particulier toute clique K_δ avec $\delta = |V(H)|$ sommets. [*Question*. Pourquoi ?]

Il suit que les familles de graphes denses nul part sont plus grandes que les familles sans mineur fixé, c'est-à-dire des graphes excluant un certain mineur fixé.

Beaucoup de graphes entrent dans la catégorie des graphes denses nul part. Citons les graphes de largeur arborescente bornée, les graphes de genre borné, les graphes sans mineurs fixés, sans mineurs topologiques fixés, ... sont denses nul part. Mais aussi les graphes de degré maximum borné qui pourtant ne sont pas toujours sans mineur fixé. [*Question*. Pourquoi ? Construire un exemple.]

Mais tous les graphes peu denses, c'est-à-dire avec $O(n)$ arêtes, ne sont pas denses nul part. (En quelque sorte, il y a des graphes peu denses qui sont denses quelque part !) Comme contre-exemples, citons les graphes d'*arboricité*[54] bornée et les graphes d-dégénérés (voir définition 2.2). Par exemple, le graphe composé d'une clique de δ sommets où chaque arête est subdivisée en deux autres, cf. la figure 2.51 où $\delta = 5$. Un tel graphe possède $n = \delta + \binom{\delta}{2} = \Theta(\delta^2)$ sommets et ses arêtes peuvent être partitionnées en deux forêts. Il a donc au plus $2(n-1)$ arêtes, il est donc peu dense. Néanmoins il n'est pas dense nul part puisqu'il contient une clique de taille $\delta = \Omega(\sqrt{n})$ comme mineur de profondeur 1.

[*Exercice*. Montrez que tout graphe où chaque arête est subdivisée en deux autres, il est possible de partitionner ses arêtes en deux forêts.]

[*Exercice*. Montrez que le *Graph Minor Theorem* 2.7 n'est pas vrai pour les familles closes par mineurs topologiques ou mineurs topologiques de profondeur ρ.]

53. Un graphe K est un sur-graphe de H si H est un sous-graphe de K.
54. Graphe dont les arêtes peuvent être partitionnées en un nombre bornée de forêts.

2.12.3 Performances

Le graphe *puissance*, noté G^p, est le graphe G dans lequel ont été ajouté toutes les arêtes entre sommets à distance au plus p dans G. Voir la figure 2.52. Il est a noter que le carré des graphes denses nul part ne forme pas forcément une famille dense nul part. Par exemple, le carré d'un arbre peut contenir des cliques de grande taille comme sous-graphe (en fait dès que le graphe contient un sommet de grand degré) et donc comme mineur de profondeur nulle.

Remarque. Si M est la matrice d'adjacence de G, alors $M^p = M \times \cdots \times M$ est la matrice où l'entrée $M^p[i,j]$ est le nombre de chemins (pas forcément simple) de longueur exactement p entre les sommets d'indice i et j de G. On peut donc déduire G^p à partir des entrées non nulles de $M + M^2 + \cdots + M^p$.

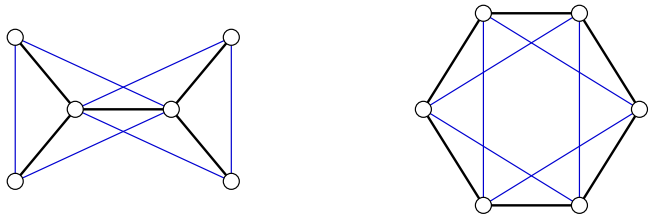


FIGURE 2.52 – Deux graphes (arêtes noires) et leurs carrés (arêtes noires et bleues). Comme ils sont de diamètre trois, leurs cubes sont des cliques.

Théorème 2.14 *L'algorithme Progress résout* ENSEMBLE DOMINANT À DISTANCE *r de taille k en temps* $k^{O(1)} \cdot (n+m) + k^{O(k)}$ *pour les graphes à n sommets et m arêtes qui sont une puissance de graphes denses nul part.*

Un algorithme similaire permet de résoudre ENSEMBLE INDÉPENDANT À DISTANCE r de taille k avec la même complexité sur les mêmes graphes. Les termes $k^{O(1)}$ et $k^{O(k)}$ cachent les constantes r, δ, ρ.

La preuve ne sera pas détaillée, car elle nécessite un travail préliminaire assez important sur les graphes nul part denses. Mais elle s'appuie sur deux points permettant d'améliorer la proposition 2.3. D'une part le nombre de profiles différents est $|R| \leqslant |W|^{1+\varepsilon}$, pour un certain $\varepsilon = \varepsilon(\rho, \delta) > 0$, au lieu de $2^{|W|}$ comme le cas général. Et d'autre part, le nombre d'itérations est polynomial en k, soit $|W| \leqslant k^{O(1)}$. Du coup $|R|^k = k^{O(k)}$ et la complexité de l'étape 2a pour la recherche de S, dans l'équation (2.3), devient :

$$\begin{aligned} O\left(|W| \cdot (n+m) + |W| \cdot |R|^k\right) &= O\left(k^{O(1)} \cdot (n+m) + k^{O(1)} \cdot k^{O(k)}\right) \\ &= O\left(k^{O(1)} \cdot (n+m) + k^{O(k)}\right). \end{aligned}$$

On obtient la complexité de l'algorithme en multipliant cette complexité par le nombre d'itérations qui est en $k^{O(1)}$, ce qui donne bien $k^{O(1)} \cdot (n+m) + k^{O(k)}$ comme annoncé dans le théorème 2.14.

2.13 Technique de coloration *(color coding)*

Pour terminer ce chapitre, nous allons illustrer une technique classique permettant d'améliorer certains algorithmes FPT, en particulier pour le problème CHEMIN DE TAILLE k déjà discuté au paragraphe 2.8.5.

L'approche naïve consiste à tester toutes les chaînes de taille k pour ne garder que celles qui forment des chemins simples. Le problème est qu'il y a beaucoup de chaînes et peu de chemins simples. On passe donc son temps à tester des cas inintéressants.

L'idée est de colorier, pas forcément proprement, les sommets du graphe en k couleurs et de restreindre la recherche seulement aux chemins utilisant k couleurs différentes. Comme on va le voir, cette recherche peut être faite plus rapidement que la version sans couleur. Le problème sur le graphe coloré devenant plus contraint, il est alors plus facile de tester tous les chemins avec k couleurs différentes que toutes les chaînes de taille k. (On va passer en gros de n^k tests à $2^k n$ tests.) La contre-partie est qu'une mauvaise coloration peut faire qu'on rate le (ou les) chemins de taille k. Heureusement, le hasard fait bien les choses.

Proposition 2.4 *Soit $c\colon V \to \{1,\dots,k\}$ la coloration d'un ensemble V où chaque élément, indépendamment, est aléatoirement et uniformément colorié dans $\{1,\dots,k\}$. Pour tout $X \subseteq V$ de taille k, la probabilité que X soit pleinement colorié, c'est-à-dire colorié avec des couleurs deux à deux différentes, est $> e^{-k}$.*

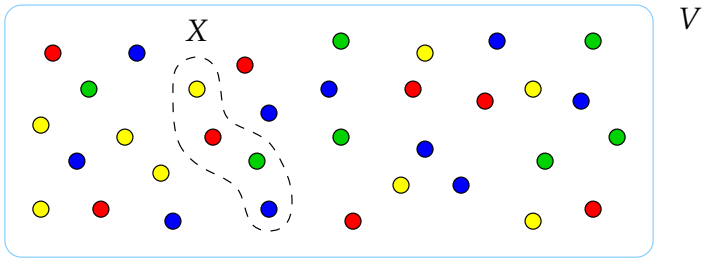


FIGURE 2.53 – Coloration aléatoire de V en $k = 4$ couleurs. La probabilité qu'un sous-ensemble donné X soit pleinement colorié est un peu moins de 10%, mais est indépendante de $|V|$. En répétant le tirage aléatoire, X va finir par être pleinement colorié avec une probabilité proche de 100%.

Preuve. La probabilité p recherchée est le nombre de colorations rendant X pleinement colorié divisé par le nombre total de colorations sur V. Le nombre total de colorations est k^n où $n = |V|$. Celles qui rendent X pleinement colorié sont obtenues en coloriant arbitrairement X de taille k par des couleurs différentes, soit $k!$ possibilités, et en coloriant arbitrairement les autres éléments de V, soit $k^{|V|-|X|} = k^{n-k}$ possibilités. Au total, et en utilisant le fait que $k! > (k/e)^k$,

$$p = \frac{k! \cdot k^{n-k}}{k^n} = k! \cdot k^{-k} > \left(\frac{k}{e}\right)^k k^{-k} = e^{-k}\,.$$

□

Il faut maintenant montrer comment déterminer rapidement dans un graphe colorié s'il existe un chemin de taille k pleinement colorié. En utilisant la programmation dynamique, il vient :

Lemme 2.8 *En temps $2^k n^{O(1)}$ il est possible de déterminer si un graphe à n sommets k-coloriés (pas forcément proprement) possède ou non un chemin de taille k pleinement colorié.*

Preuve. Soit $c\colon V \to \{1,\dots,k\}$ la coloration d'un graphe G avec $V(G) = V$.

Pour tout sous-ensemble S de couleurs et sommet u, on définit chemin(u,S) le prédicat indiquant s'il existe ou non un chemin d'extrémité u et de taille $|S|$, pleinement colorié par les couleurs de S.

L'objectif est de déterminer s'il existe un sommet u tel que chemin$(u,\{1,\dots,k\}) =$ Vrai. On a la récurrence suivante :

$$\text{chemin}(u,S) \;=\; \begin{cases} \textsc{Faux} & \text{si } c(u) \notin S \\ \textsc{Vrai} & \text{si } S = \{c(u)\} \\ \bigvee_{v\in N(u)} \text{chemin}(v, S \setminus \{c(u)\}) & \text{sinon} \end{cases}$$

Les deux premiers cas sont évidents. Pour le troisième, il est clair que u est l'extrémité d'un chemin pleinement colorié par S si et seulement si au moins un de ces voisins est l'extrémité d'un chemin pleinement colorié par S sans la couleur de u.

Donc pour calculer chemin(u,S) il faut avoir préalablement calculer chemin(v,S') pour tous les sous-ensembles $S' \subset S$ de taille inférieure et tout sommet v. Il y a au plus $n2^{|S|} \leqslant n2^k$ couples (v,S') différents au total qui peuvent être stockées dans une table (mémoïsation). La recherche de l'entrée (v,S') dans les valeurs déjà calculées peut se faire en temps $O(\log(n2^k)) = O(k + \log n)$ en implémentant la table par un arbre de cherche équilibré (qui a donc une hauteur logarithmique en le nombre d'entrées). Le calcul de chemin(u,S) à proprement parlé nécessite la combinaison de $\deg(u)$ valeurs à cause du « $\bigvee_{v\in N(u)}$ » une fois le calcul de $S \setminus \{c(u)\}$ effectué, qui lui prend un temps $O(|S|) = O(k)$.

Au total, le calcul de tous les chemin(u,S), en majorant k par n, se somme en :

$$\sum_{u\in V} \sum_{S\subseteq\{1,\dots,k\}} (O(k) + \deg(u)\cdot O(k+\log n)) \;=\; \sum_{S\subseteq\{1,\dots,k\}} \sum_{u\in V} O(\deg(u)\cdot n) \;=\; O(nm\cdot 2^k)\,.$$

□

En combinant les deux résultats précédents on obtient :

Théorème 2.15 *On peut résoudre* Chemin de taille k *pour un graphe à n sommets par un algorithme probabiliste en temps $(2e)^k n^{O(1)}$. Cet algorithme renvoie un tel chemin, s'il existe, avec une probabilité supérieure à 99%.*

L'exposant exact du terme $n^{O(1)}$ dépend de la façon d'implémenter la procédure de vérification dans le lemme 2.8, mais on peut l'estimer à n^3 environ. On peut aussi modifier légèrement la procédure pour qu'elle retourne le chemin et pas seulement la valeur de vérité.

Preuve. L'algorithme est le suivant :

Algorithme RandLongestPath(G,k)

Répéter au plus $t := \lceil 5e^k \rceil$ fois :

(a) Colorier les sommets de G en k couleurs, aléatoirement et uniformément.

(b) S'il existe un chemin pleinement colorié de taille k, le renvoyer.

La complexité de l'algorithme est $t \cdot 2^k n^{O(1)} = 5e^k \cdot 2^k n^{O(1)} = (2e)^k n^{O(1)}$ comme annoncée.

Clairement, l'algorithme ne retourne pas de chemin si le graphe ne possède pas de chemin de taille k. Supposons que le graphe possède effectivement un chemin de taille k. L'objectif est de minorer la probabilité de succès, c'est-à-dire que l'algorithme détecte un tel chemin.

La probabilité que l'algorithme échoue est donnée par la probabilité d'échouer t fois de suite au test (2). Si on note p la probabilité de réussir le test (2), alors la probabilité d'échec de l'algorithme est $(1-p)^t$.

Fait 2.3 *Pour tout* $0 < \alpha \leqslant 1/2$ *et* $\beta > 0$, $(1-\alpha)^\beta < e^{-\alpha\beta}$.

Donc la probabilité que l'algorithme échoue est $(1-p)^t < e^{-pt}$. D'après la proposition 2.4, $p > e^{-k}$. Notons que $5/p < 5e^k \leqslant \lceil 5e^k \rceil = t$ (d'après l'algorithme), soit $5 < pt$. Et donc $e^{-pt} < e^{-5}$.

Ainsi, si le graphe possède un chemin de taille k, l'algorithme le trouvera avec une probabilité d'au moins $1 - e^{-5} \approx 0.9932 > 99\%$. □

Il existe une technique encore plus efficace que *color coding* qui a été développée plus récemment par [BHKK17] : *narrow sieves*. Elle a mené à l'algorithme (probabiliste) le plus rapide connus pour le problème Chemin de taille de taille k (mais pas que) et qui a pour complexité $1.657^k n^{O(1)}$. Le meilleure algorithme déterministe pour ce problème a une complexité en $2^k n^{O(1)}$ [Wil09]. Notons que pour $k = n$, on en déduit des algorithmes pour le problème Hamiltonisme ayant respectivement des complexités en $O(1.657^n)$ (pour un algorithme probabiliste) et $O(2^n)$ (pour un algorithme déterministe). [*Question*. Quelle serait la complexité d'un algorithme naïf pour Hamiltonisme ?]

Des algorithmes plus efficaces en $2^{O(\sqrt{k})} n^{O(1)}$ existent dans le cas des graphes excluant un mineur H donné [DFT08], et en particulier pour les graphes planaires.

La détection de *sentier* (*trail* en Anglais), c'est-à-dire un chemin induit qui n'est pas un plus court chemin [55] est cependant polynomial. En effet, savoir s'il existe un sentier entre deux sommets donnés (en donner un ou dire qu'il n'en existe pas) peut-être résolu en $n^{2\omega} \cdot \text{polylog}(n) < n^{4.75}$ où [8] $\omega < 2.3729$, cf. [CL21]. Ce n'est aussi que très récemment [CL22] que le problème de trouver le plus court cycle induit de longueur paire a été montré polynomial, avec un premier algorithme en $O(n^{31})$. Pour le plus long cycle induit de longueur paire, c'est NP-difficile (idem pour la longueur impaire). [*Exercice*. Proposez une réduction.] Alors qu'il est polynomial de savoir si un graphe possède un cycle induit de longueur paire (et idem pour la longueur impaire), c'est NP-difficile de savoir s'il y a un cycle induit de longueur impaire passant par un sommet donné, cf. [CSS20].

Bibliographie

[ACHS21] T. Abrishami, M. Chudnovsky, S. Hajebi, and S. T. Spirkl, *Induced subgraphs and tree decompositions III. three-path-configurations and logarithmic treewidth*, Tech. Rep. 2109.013104v1 [math.CO], arXiv, September 2021.

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM Journal on Algebraic and Discrete Methods, 8 (1987), pp. 277–284. doi : 10.1137/0608024.

[AFF+05] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege, *A refined search tree technique for Dominating Set on planar graphs*, Journal of Computer and System Sciences, 71 (2005), pp. 385–405. doi : 10.1016/j.jcss.2004.03.007.

[AGK08] I. Adler, M. Grohe, and S. Kreutzer, *Computing excluded minors*, in 19th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2008, pp. 641–650. https://dl.acm.org/doi/10.5555/1347082.1347153.

[AK97] P. Alimonti and V. Kann, *Hardness of approximating problems on cubic graphs*, in 3rd Italian Conference on Algorithms and Complexity (CIAC), vol. 1203 of Lecture Notes in Computer Science, Springer, March 1997, pp. 288–298. doi : 10.1007/3-540-62592-5_80.

[AKFLS07] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters, *Crown structures for vertex cover kernelization*, Theory of Computing Systems, 41 (2007), pp. 411–430. doi : 10.1007/s00224-007-1328-0.

[Ami01] E. Amir, *Efficient approximation for triangulation of minimum treewidth*, in 17th Conference on Uncertainty in Artificial Intelligence (UAI), Morgan Kaufmann, August 2001, pp. 7–15.

55. Rappelons qu'entre toute paire de sommet u, v d'un graphe connexe il existe toujours un chemin induit : le plus court chemin entre u et v.

[AP87] S. Arnborg and A. Proskurowski, *Characterization and recognition of partial 3-trees*, SIAM Journal on Algebraic and Discrete Methods, 7 (1987), pp. 305–314. doi : 10.1137/0607033.

[APC90] S. Arnborg, A. Proskurowski, and D. G. Corneil, *Forbidden minors characterization of partial 3-trees*, Discrete Mathematics, 80 (1990), pp. 1–19. doi : 10.1016/0012-365X(90)90292-P.

[Arc81] D. Archdeacon, *A Kuratowski theorem for the projective plane*, Journal of Graph Theory, 5 (1981), pp. 243–246. doi : 10.1002/jgt.3190050305.

[BDD+16] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, and D. Lokshtanov, *A $c^k n$ 5-approximation algorithm for treewidth*, SIAM Journal on Computing, 45 (2016), pp. 317–378. doi : 10.1137/130947374.

[BF21a] M. Belbasi and M. Fürer, *Finding all leftmost separators of size $\leqslant k$*, in 15th International Conference on Combinatorial Optimization and Applications (COCOA), D.-Z. Du, D. Du, C. Wu, and D. Xu, eds., vol. 13135 of Lecture Notes in Computer Science, Springer, December 2021, pp. 273–287. doi : 10.1007/978-3-030-92681-6_23.

[BF21b] M. Belbasi and M. Fürer, *An improvement of Reed's treewidth approximation*, in 15th International Conference and Workshops on Algorithms and Computation (WALCOM), R. Uehara, S.-H. Hong, and S. C. Nandy, eds., vol. 12635 of Lecture Notes in Computer Science, Springer, February 2021, pp. 166–181. doi : 10.1007/978-3-030-68211-8_14.

[BFL+09] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos, *(Meta) Kernelization*, in 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 2009, pp. 629–638. doi : 10.1109/FOCS.2009.46.

[BHKK17] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, *Narrow sieves for parameterized paths and packings*, Journal of Computer and System Sciences, 87 (2017), pp. 119–139. doi : 10.1016/j.jcss.2017.03.003.

[BK10] H. L. Bodlaender and A. M. Koster, *Treewidth computations I. upper bounds*, Information and Computation, 208 (2010), pp. 259–275. doi : 10.1016/j.ic.2009.03.008.

[BK11] H. L. Bodlaender and A. M. Koster, *Treewidth computations II. lower bounds*, Information and Computation, 209 (2011), pp. 1103–1119. doi : 10.1016/j.ic.2011.04.003.

[BKMT04] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca, *On treewidth approximations*, Discrete Applied Mathematics, 136 (2004), pp. 183–196. doi : 10.1016/S0166-218X(03)00440-2.

[Blu84] N. Blum, *A boolean function requiring 3n network size*, Theoretical Computer Science, 28 (1984), pp. 337–345. doi : 10.1016/0304-3975(83)90029-4.

[Bod93] H. L. Bodlaender, *A tourist guide through treewidth*, Acta Cybernetica, 11 (1993), pp. 1–21.

[Bod96] H. L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM Journal on Computing, 25 (1996), pp. 1305–1317. doi : 10.1137/S0097539793251219.

[CE12] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic*, vol. 138 of Encyclopedia of Mathematics and Its Applications, Cambridge University Press, July 2012. isbn : 0521898331, 9780521898331.

[CFKX05] J. Chen, H. Fernau, I. A. Kanj, and G. Xia, *Parametric duality and kernelization : Lower bounds and upper bounds on kernel size*, in 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 3404 of Lecture Notes in Computer Science, Springer, February 2005, pp. 269–280. doi : 10.1007/978-3-540-31856-9_22.

[CG83] J. H. Conway and C. M. Gordon, *Knots and links in spatial graphs*, Journal of Graph Theory, 7 (1983), pp. 445–453. doi : 10.1002/jgt.3190070410.

[CKX06] J. Chen, I. A. Kanj, and G. Xia, *Improved parameterized upper bounds for Vertex Cover*, in 31st International Symposium on Mathematical Foundations of Computer Science (MFCS), R. Kráľovič and P. Urzyczyn, eds., vol. 4162 of Lecture Notes in Computer Science, Springer, August 2006, pp. 238–249. doi : 10.1007/11821069_21.

[CL21] Y.-C. Chiu and H.-I. Lu, *Blazing a trail via matrix multiplications : A faster algorithm for non-shortest induced paths*, Tech. Rep. 2109.15268v1 [cs.DS], arXiv, September 2021.

[CL22] H.-T. Cheong and H.-I. Lu, *Finding a shortest even hole in polynomial time*, Journal of Graph Theory, 99 (2022), pp. 425–434. doi : 10.1002/jgt.22748.

[Cou90] B. Courcelle, *The monadic second-order logic of graphs. I. Recognizable sets of finite graphs*, Information and Computation, 85 (1990), pp. 12–75. doi : 10.1016/0890-5401(90)90043-H.

[CSS20] M. Chudnovsky, A. Scott, and P. D. Seymour, *Finding a shortest odd hole*, Tech. Rep. 2004.11874v1 [math.CO], arXiv, April 2020.

[CT19] C. Chekuri and Z. Tan, *Towards tight(er) bounds for the excluded grid theorem*, in 30th Symposium on Discrete Algorithms (SODA), ACM Press, January 2019, pp. 1445–1464. doi : 10.1137/1.9781611975482.88.

[DEW15] V. Dujmović, D. Eppstein, and D. R. Wood, *Genus, treewidth, and local crossing number*, in 23rd International Symposium on Graph Drawing and Network Visualization (GD), vol. 9411 of Lecture Notes in Computer Science, Springer, September 2015, pp. 87–98. doi : 10.1007/978-3-319-27261-0_8.

[DFT08] F. Dorn, F. V. Fomin, and D. M. Thilikos, *Catalan structures and dynamic programming in H-minor-free graphs*, in 19th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2008, pp. 631–640. https://dl.acm.org/doi/10.5555/1347082.1347152.

[DHV19] G. Ducoffe, M. Habib, and L. Viennot, *Diameter computation on H-minor free graphs and graphs of bounded (distance) VC-dimension*, Tech. Rep. 1907.04385v2 [cs.DS], arXiv, October 2019.

[DK07] A. P. Dow and R. E. Korf, *Best-first search for treewidth*, in 22nd National Conference on Artificial Intelligence (AAAI), vol. 2, AAAI Press, July 2007, pp. 1146–1151.

[DMW17] V. Dujmović, P. Morin, and D. R. Wood, *Layered separators in minor-closed families with applications*, Journal of Combinatorial Theory, Series B, 127 (2017), pp. 111–147. doi : 10.1016/j.jctb.2017.05.006.

[DN19] Z. Dvořák and S. Norin, *Treewidth of graphs with balanced separations*, Journal of Combinatorial Theory, Series B, 137 (2019), pp. 137–144. doi : 10.1016/j.jctb.2018.12.007.

[Dor10] F. Dorn, *Dynamic programming and planarity : Improved tree-decomposition based algorithms*, Discrete Applied Mathematics, 158 (2010), pp. 800–808. doi : 10.1016/j.dam.2009.10.011.

[EJM12] K. Elbassioni, S. Jelić, and D. Matijevicć, *The relation of connected set cover and group steiner tree*, Theoretical Computer Science, 438 (2012), pp. 96–101. doi : 10.1016/j.tcs.2012.02.035.

[FG06] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer, 2006. isbn : 978-3-540-29952-3, 978-3-540-29953-0. doi : 10.1007/3-540-29953-X.

[FGHK16] M. G. Find, A. Golovnev, E. A. Hirsch, and A. S. Kulikov, *A better-than-3n lower bound for the circuit complexity of an explicit function*, in 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 2016, pp. 89–98. doi : 10.1109/FOCS.2016.19.

[FGK09] F. V. Fomin, F. Grandoni, and D. Kratsch, *A measure & conquer approach for the analysis of exact algorithms*, Journal of the ACM, 56 (2009), pp. Article No. 25, pp. 1–32. doi : 10.1145/1552285.1552286.

[FHL05] U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum-weight vertex separators*, in 37th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 2005, pp. 563–572. doi : 10.1145/1060590.1060674.

[FL89] M. R. Fellows and M. A. Langston, *On search, decision and the efficiency of polynomial-time algorithms*, in 21st Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 1989, pp. 501–512. doi : 10.1145/73007.73055.

[FLS+18] F. V. Fomin, D. Lokshtanov, S. Saurabh, M. Pilipczuk, and M. Wrochna, *Fully polynomial-time parameterized computations for graphs and matrices of low treewidth*, ACM Transactions on Algorithms, 14 (2018), pp. Article No. 34, pp. 1–45. doi : 10.1145/3186898.

[FPST18] G. Fabiański, M. Pilipczuk, S. Siebertz, and S. Toruńczyk, *Progressive algorithms for domination and independence*, Tech. Rep. 1811.06799v1 [cs.LO], arXiv, November 2018.

[FT06] F. V. Fomin and D. M. Thilikos, *Dominating sets in planar graphs : Branch-width and exponential speed-up*, SIAM Journal on Computing, 36 (2006), pp. 281–306. doi : 10.1137/S0097539702419649.

[FTKV08] F. V. Fomin, I. Todinca, D. Kratsch, and Y. Villanger, *Exact algorithms for treewidth and minimum fill-in*, SIAM Journal on Computing, 38 (2008), pp. 1058–1079. doi : 10.1137/050643350.

[FTV15] F. V. Fomin, , I. Todinca, and Y. Villanger, *Large induced sub-graphs via triangulations and CMSO*, SIAM Journal on Computing, 44 (2015), pp. 54–87. doi : 10.1137/140964801.

[Gav23] C. Gavoille, *Techniques algorithmiques et programmation*, 2023. http://dept-info.labri.fr/~gavoille/UE-TAP/cours.pdf. Notes de cours.

[GH13] W. Goddard and M. A. Henning, *Independent domination in graphs : A survey and recent results*, Discrete Mathematics, 313 (2013), pp. 839–854. doi : 10.1016/j.disc.2012.11.031.

[GHW79] H. H. Glover, J. P. Huneke, and C. S. Wang, *103 graphs that are irreducible for the projective plane*, Journal of Combinatorial Theory, Series B, 27 (1979), pp. 332–370. doi : 10.1016/0095-8956(79)90022-4.

[GK98] S. Guha and S. Khuller, *Approximation algorithms for connected dominating sets*, Algorithmica, 20 (1998), pp. 374–387. doi : 10.1007/PL00009201.

[GKR13] M. Grohe, K.-i. Kawarabayashi, and B. A. Reed, *A simple algorithm for the graph minor decomposition – Logic meets structural graph theory –*, in 24th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2013, pp. 414–431. doi : 10.1137/1.9781611973105.30.

[GL06] M. Grantson and C. Levcopoulos, *Covering a set of points with a minimum number of lines*, in 6th Italian Conference on Algorithms and Complexity (CIAC), vol. 3998 of Lecture Notes in Computer Science, Springer, May 2006, pp. 6–17. doi : 10.1007/11758471_4.

[GMC09] A. Gagarin, W. J. Myrvold, and J. Chambers, *The obstructions for toroidal graphs with no $K_{3,3}$'s*, Discrete Mathematics, 309 (2009), pp. 3625–3631. doi : 10.1016/j.disc.2007.12.075.

[GMT02] J. Gustedt, O. A. Mæhle, and J. A. Telle, *The treewidth of java programs*, in 4th International Workshop on Algorithm Engineering and Experiments

(ALENEX), vol. 2409 of Lecture Notes in Computer Science, Springer, January 2002, pp. 86–97. DOI : 10.1007/3-540-45643-0_7.

[GT12] Q.-P. Gu and H. Tamaki, *Improved bounds on the planar branchwidth with respect to the largest grid minor size*, Algorithmica, 64 (2012), pp. 416–453. DOI : 10.1007/s00453-012-9627-5.

[GUM10] A. Grigoriev, N. Usotskaya, and B. Marchal, *On planar graphs with large tree-width and small grid minors*, Electronic Notes in Discrete Mathematics, 32 (2010), pp. 35–42. DOI : 10.1016/j.endm.2009.02.006.

[Hal16] J. T. Halseth, *A 43k kernel for planar dominating set using computer-aided reduction rule discovery*, master thesis, University of Bergen, February 2016.

[Hås98] J. Håstad, *The shrinkage exponent of de morgan formulae is 2*, SIAM Journal on Computing, 27 (1998), pp. 48–64. DOI : 10.1137/S0097539794261556.

[HW17] D. J. Harveyn and D. R. Wood, *Parameters tied to treewidth*, Journal of Graph Theory, 84 (2017), pp. 364–385. DOI : 10.1002/jgt.22030.

[IM02] K. Iwama and H. Morizumi, *An explicit lower bound of* $5n-o(n)$ *for boolean circuits*, in 27th International Symposium on Mathematical Foundations of Computer Science (MFCS), vol. 2420 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 353–364. DOI : 10.1007/3-540-45687-2_29.

[Joh87] D. S. Johnson, *The NP-completeness column : An ongoing guide – The many faces of polynomial time*, Journal of Algorithms, 8 (1987), pp. 285–303. DOI : 10.1016/0196-6774(87)90043-5.

[Khu02] S. Khuller, *The vertex cover problem*, ACM SIGACT News - Algorithms Column, 33 (2002), pp. 31–33. DOI : 10.1145/564585.564598.

[KKM09] K.-i. Kawarabayashi, S. Kreutzer, and B. Mohar, *Linkless and flat embeddings in* 3*-space in quadratic time*, Tech. Rep. 1070, University of Ljubljana, Slovenia, January 2009.

[KKR12] K.-i. Kawarabayashi, Y. Kobayashi, and B. A. Reed, *The disjoint paths problem in quadratic time*, Journal of Combinatorial Theory, Series B, 102 (2012), pp. 424–435. DOI : 10.1016/j.jctb.2011.07.004.

[KL23] T. Korhonen and D. Lokshtanov, *An improved parameterized algorithm for treewidth*, in 29th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, June 2023, pp. 528–541. DOI : 10.1145/3564246.3585245.

[Kor22] T. Korhonen, *A single-exponential time 2-approximation algorithm for treewidth*, in 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, February 2022, pp. 184–192. DOI : 10.1109/FOCS52979.2021.00026.

[KT11] A. Koutsonas and D. M. Thilikos, *Planar feedback vertex set and face cover : Combinatorial bounds and subexponential algorithms*, Algorithmica, 60 (2011), pp. 987–1003. doi : 10.1007/s00453-010-9390-4.

[KT16] F. Kammer and T. Tholey, *Approximate tree decompositions of planar graphs in linear time*, Tech. Rep. 1104.2275v5 [cs.DS], arXiv, May 2016.

[LA91] J. Lagergren and S. Arnborg, *Finding minimal forbidden minors using a finite congruence*, in 18th International Colloquium on Automata, Languages and Programming (ICALP), vol. 510 of Lecture Notes in Computer Science, Springer, July 1991, pp. 532–543. doi : 10.1007/3-540-54233-7_161.

[Lag96] J. Lagergren, *Efficient parallel algorithms for graphs of bounded tree-width*, Journal of Algorithms, 20 (1996), pp. 20–44. doi : 10.1006/jagm.1996.0002.

[LMS11] D. Lokshtanov, D. Marx, and S. Saurabh, *Known algorithms on graphs of bounded treewidth are probably optimal*, in 22nd Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2011, pp. 777–789. doi : 10.1137/1.9781611973082.61.

[LPL15] C.-H. Liu, S.-H. Poon, and J.-Y. Lin, *Independent dominating set problem revisited*, Theoretical Computer Science, 562 (2015), pp. 1–22. doi : 10.1016/j.tcs.2014.09.001.

[Luc07] B. Lucena, *Achievable sets, brambles, and sparse treewidth obstructions*, Discrete Applied Mathematics, 155 (2007), pp. 1055–1065. doi : 10.1016/j.dam.2006.11.006.

[MT91] J. Matoušek and R. Thomas, *Finding tree-decompositions of graphs*, Journal of Algorithms, 12 (1991), pp. 1–22. doi : 10.1016/0196-6774(91)90020-Y.

[Neč66] è. I. Nečiporuk, *On a boolean function*, Doklady Akad. Nauk SSSR, 169 (1966), pp. 765–766.

[Nie06] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, 2006.

[NOdM08a] J. Nešetřil and P. Ossona de Mendez, *Grad and classes with bounded expansion I. Decompositions*, European Journal of Combinatorics, 29 (2008), pp. 760–776. doi : 10.1016/j.ejc.2006.07.013.

[NOdM08b] J. Nešetřil and P. Ossona de Mendez, *Grad and classes with bounded expansion II. Algorithmic aspects*, European Journal of Combinatorics, 29 (2008), pp. 777–791. doi : 10.1016/j.ejc.2006.07.014.

[NOdMW12] J. Nešetřil, P. Ossona de Mendez, and D. R. Wood, *Characterisations and examples of graph classes with bounded expansion*, European Journal of Combinatorics, 33 (2012), pp. 350–373. doi : 10.1016/j.ejc.2011.09.008.

[NP85] J. Nešetřil and S. Poljak, *On the complexity of subgraph problem*, Commentationes Mathematicae Universatis Carolinae, 26 (1985), pp. 415–419.

[PRS12] G. Philip, V. Raman, and S. Sikdar, *Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond*, ACM Transactions on Algorithms, 9 (2012), pp. Article No. 11, pp. 1–23. doi : 10.1145/2390176.2390187.

[Ree92] B. A. Reed, *Finding approximate separators and computing treewidth quickly*, in 24th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, 1992, pp. 221–228. doi : 10.1145/129712.129734.

[RL08] B. A. Reed and Z. Li, *Optimization and recognition for k_5-minor free graphs in linear time*, in 8th Latin American Symposium on Theoretical Informatics (LATIN), vol. 4957 of Lecture Notes in Computer Science, April 2008, pp. 206–215. doi : 10.1007/978-3-540-78773-0_18.

[RS84] N. Robertson and P. D. Seymour, *Graph width and well-quasi-ordering : a survey*, in Progress in Graph Theory, A. J. Bondy and U. Murty, eds., Academic Press, 1984, pp. 399–406.

[RS91] N. Robertson and P. D. Seymour, *Graph minors. X. Obstructions to tree-decompositions*, Journal of Combinatorial Theory, Series B, 52 (1991), pp. 153–190. doi : 10.1016/0095-8956(91)90061-N.

[RS95] N. Robertson and P. D. Seymour, *Graph minors. XIII. The disjoint paths problem*, Journal of Combinatorial Theory, Series B, 63 (1995), pp. 65–110. doi : 10.1006/jctb.1995.1006.

[RS04] N. Robertson and P. D. Seymour, *Graph minors. XX. Wagner's conjecture*, Journal of Combinatorial Theory, Series B, 92 (2004), pp. 325–357. doi : 10.1016/j.jctb.2004.08.001.

[RST94] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, Journal of Combinatorial Theory, Series B, 62 (1994), pp. 323–348. doi : 10.1006/jctb.1994.1073.

[RST95] N. Robertson, P. D. Seymour, and R. Thomas, *Sachs' linkless embedding conjecture*, Journal of Combinatorial Theory, Series B, 64 (1995), pp. 185–227. doi : 10.1006/jctb.1995.1032.

[RZ11] W. Ren and Q. Zhang, *A note on 'algorithms for connected set cover problem and fault-tolerant connected set cover problem'*, Theoretical Computer Science, 412 (2011), pp. 6451–6454. doi : 10.1016/j.tcs.2011.07.008.

[Sac81] H. Sachs, *On spatial representation of finite graphs*, in Finite and Infinite Sets, A. Hajnal, L. Lovász, and V. T. Sós, eds., vol. 37 of Colloquia Mathematica Societatis János Bolyai, North-Holland, 1981, pp. 649–662. doi : 10.1016/C2013-0-04465-9.

[Sac83] H. Sachs, *On a spatial analogue of Kuratowski's Theorem on planar graphs - an open problem*, in Graph Theory : Proceedings of a Conference held in Łagów, Poland, February 10-13, 1981, M. Borowiecki, J. W. Kennedy, and M. M. Sysło, eds., vol. 1018 of Lecture Notes in Mathematics, Springer-Verlag, 1983, pp. 230–241. DOI : 10.1007/BFb0071633.

[San93] D. P. Sanders, *Linear algorithms for graphs of tree-width at most four*, PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, August 1993. http://hdl.handle.net/1853/30061.

[Sha49] C. E. Shannon, *The synthesis of two-terminal switching circuits*, Bell Systems Technical Journal, 28 (1949), pp. 59–98.

[SST20] I. Sau, G. Stamoulis, and D. M. Thilikos, *An FPT-algorithm for recognizing k-apices of minor-closed graph classes*, in 47th International Colloquium on Automata, Languages and Programming (ICALP), A. Czumaj, A. Dawar, and E. Merelli, eds., vol. 168 of LIPIcs, July 2020, pp. 95 :1–95 :20. DOI : 10.4230/LIPIcs.ICALP.2020.95.

[Tal14] A. Tal, *Shrinkage of de morgan formulae by spectral techniques*, in 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 2014, pp. 551–560. DOI : 10.1109/FOCS.2014.65.

[Tho98] M. Thorup, *All structured programs have small tree-width and good register allocation*, Information and Computation, 142 (1998), pp. 159–181. DOI : 10.1006/inco.1997.2697.

[TUK94] A. Takahashi, S. Ueno, and Y. Kajitani, *Minimal acyclic forbidden minors for the family of graphs with bounded path-width*, Discrete Mathematics, 127 (1994), pp. 293–304. DOI : 10.1016/0012-365X(94)90092-2.

[Vik96] N. 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 (1996), pp. 1–9. DOI : 10.1006/jcss.1996.0045.

[vRBR09] J. M. van Rooij, H. L. Bodlaender, and P. Rossmanith, *Dynamic programming on tree decompositions using generalised fast subset convolution*, in 17th Annual European Symposium on Algorithms (ESA), vol. 5757 of Lecture Notes in Computer Science, Springer, September 2009, pp. 566–577. DOI : 10.1007/978-3-642-04128-0_51.

[WB15] R. R. Williams and S. R. Buss, *Limits on alternation-trading proofs for time–space lower bounds*, Computational Complexity, 24 (2015), pp. 533–600. DOI : 10.1007/s00037-015-0104-9.

[Wil09] R. R. Williams, *Finding paths of length k in $O^*(2^k)$ time*, Information Processing Letters, 109 (2009), pp. 315–318. DOI : 10.1016/j.ipl.2008.11.004.

[Wil10] R. R. Williams, *Alternation-trading proofs, linear programming, and lower bounds*, in 27th Annual Symposium on Theoretical Aspects of Compu-

ter Science (STACS), HAL Inria-00456011, March 2010, pp. 669–680. https://hal.inria.fr/inria-00456011.

[YAK14] A. Yamaguchi and K. F. Aoki-Kinoshita, *Chemical compound complexity in biological pathways*, in Quantitative Graph Theory : Mathematical Foundations and Applications, M. Dehmer and F. Emmert-Streib, eds., Discrete Mathematics and Its Applications, CRC Press, 2014, ch. 16, pp. 471–493. doi : 10.1201/b17645-17.

[ZH09] R. Zhou and E. A. Hansen, *Combining breadth-first and depth-first strategies in searching for treewidth*, in 21st International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann Publishers Inc., July 2009, pp. 640–645. http://ijcai.org/Proceedings/09/Papers/112.pdf.

CHAPITRE

3 Algorithmes d'approximation

> *Tout problème complexe a une solution simple*
> *... mais erronée.*

Sommaire

3.1 Introduction

Dans ce chapitre on s'intéresse à des problèmes d'*optimisation* plutôt qu'à des problèmes de décisions. La plupart des problèmes de décisions que l'on a considéré jusqu'à présent ont une variante d'optimisation : Couverture par sommets de taille minimum, Ensemble indépendant de taille maximum, Ensemble dominant de taille minimum, etc. Pour d'autres problèmes, la variante d'optimisation est moins claire comme les problèmes Sat ou Primalité.

Il est clair que la variante d'optimisation d'un problème (de décision) NP-complet ne possède pas d'algorithme polynomial, sauf si P=NP. En effet, on pourrait décider de l'existence d'une solution de valeur k donnée en comparant simplement k à la valeur optimale. S'il y a peu d'espoir, pour un problème difficile, de trouver un algorithme exact polynomial, il y en a en revanche des polynomiaux qui donnent des solutions approchées.

3.1.1 Définition

Pour une instance I d'un problème d'optimisation Π, on notera

- $\text{OPT}_\Pi(I)$ la valeur de la solution optimale pour l'instance I ; et
- $\mathsf{A}(I)$ la valeur de la solution produite par l'algorithme A sur l'instance I.

Parfois on notera $\text{OPT}(I)$ ou même simplement OPT lorsque Π et I sont claires d'après le contexte. Pour simplifier, on supposera toujours que le problème Π est à valeurs positives [1], c'est-à-dire que $\text{OPT}_\Pi(I) \geqslant 0$ pour toute instance I.

Un algorithme d'approximation a donc pour vocation de produire une solution de valeur « relativement proche » de l'optimal, notion que l'on définit maintenant [2].

Définition 3.1 (α-approximation) *Une* α-approximation *pour un problème d'optimisation* Π *est un algorithme polynomial* A *qui donne une solution pour toute instance* $I \in \Pi$ *telle que :*

- $\mathsf{A}(I) \leqslant \alpha \cdot \text{OPT}_\Pi(I)$ *dans le cas d'une minimisation ; et*
- $\mathsf{A}(I) \geqslant \alpha \cdot \text{OPT}_\Pi(I)$ *dans le cas d'une maximisation.*

La valeur α *est le* facteur d'approximation *de l'algorithme* A.

Complexité polynomiale et facteur d'approximation borné, sont deux propriétés désirables pour les problèmes NP. Car, premièrement, on peut toujours calculer une solution optimale en listant tous les certificats positifs possibles. C'est la méthode *brute-force* qui prend un temps exponentiel en la taille (polynomiale) des certificats. Calculer une solution seulement approchée en un temps qui serait non-polynomial a donc relativement peu d'intérêts. Et, deuxièmement, pouvoir dire quelque chose sur ce que produit l'algorithme et garantir un facteur d'approximation indépendant de la taille des instances est également un critère important. Sans cela on peut toujours calculer rapidement une solution arbitrairement loin de l'optimale. Par exemple, en la tirant au hasard.

Dans le cas d'une minimisation $\alpha \geqslant 1$ car $\text{OPT}(I) \leqslant \mathsf{A}(I) \leqslant \alpha \cdot \text{OPT}(I)$, et pour une maximisation $\alpha \leqslant 1$ car $\alpha \cdot \text{OPT}(I) \leqslant \mathsf{A}(I) \leqslant \text{OPT}(I)$. Remarquons qu'une 1-approximation est un algorithme exact polynomial. La terminologie anglaise utilise parfois (et pas toujours !) une autre convention pour les maximisations : une α-approximation doit vérifier $\mathsf{A}(I) \geqslant \text{OPT}_\Pi(I)/\alpha$ de sorte que $\alpha \geqslant 1$ pour un problème de minimisation et de maximisation.

1. Sinon on peut toujours s'intéresser au problème similaire renvoyant la valeur opposée, la valeur absolue ou une translation de la valeur, quitte à transformer une maximisation en minimisation (ou le contraire).

2. La définition peut varier suivant le sens précis que l'on veut donner à « relativement proche ». Parfois on souhaite des approximations à un facteur additif près plutôt que multiplicatif. Parfois, on impose le facteur d'approximation seulement pour les instances suffisamment grandes, puisque pour les très petites, un algorithme exponentiel peut en venir à bout en un temps raisonnable (en fait en temps constant si la taille était constante).

Une *instance critique* pour une α-approximation A est une instance I de taille arbitraire pour laquelle $\mathsf{A}(I) = \alpha \cdot \textsc{opt}(I)$. Cela revient à dire que l'analyse du facteur d'approximation α de A est la plus fine possible. Les instances critiques sont importantes pour comprendre le comportement de l'algorithme, pour voir ses défauts qu'on peut éventuellement essayer de corriger.

Il est important que l'égalité précédente se produise pour des instances arbitrairement grandes. En effet, supposons que les seules instances critiques de A qui font que $\alpha \neq 1$ sont de taille constante, disons bornée par n_0. Alors on pourrait construire un autre algorithme qui donnerait la solution optimale, en la cherchant de manière exhaustive, pour les instances de taille $\leqslant n_0$ en temps $T(n_0) = O(1)$ constant, et qui appliquerait A pour toutes les autres instances. Il en résulterait un algorithme polynomial exact pour toutes les instances, soit une 1-approximation. Des instances critiques de taille bornée ne sont donc pas très pertinentes.

Il est important de bien faire la différence entre un algorithme d'approximation et une heuristique. Dans le premier cas, il s'agit d'un algorithme de complexité polynomiale avec un facteur d'approximation garanti. Une heuristique ne satisfait pas forcément ces deux critères. Citons comme exemple l'heuristique suivante (cf. figure 3.1) utilisée pour calculer une tournée de coût minimum pour le problème du Voyageur de commerce (dont on va reparler ci-après) dans un graphe complet valué (G, ω) :

Algorithme Swap(G, ω)

1. Choisir une tournée quelconque de G.
2. Tant qu'il existe deux arêtes prise sur la tournée dont l'échange diminue strictement le coût, réaliser l'échange.

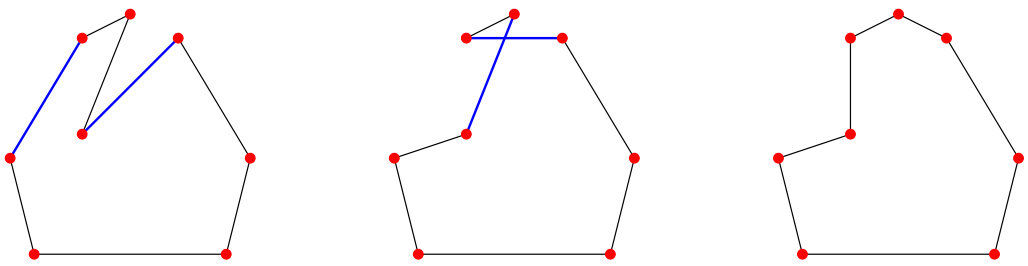


Figure 3.1 – Amélioration progressive de la tournée par applications de Swap.

Il a été montré que la boucle « Tant que » de cette heuristique peut être répétée jusqu'à $2^{n/8-O(1)}$ fois [ERV07]. Ce n'est donc pas un algorithme polynomial. De plus, il y a aucune garantie sur la qualité de l'approximation. Même dans le cas euclidien (où les sommets sont des points de $\mathbb{R}^2$ et le poids des arêtes la distance euclidienne entre les points), il est possible que la solution trouvée soit éloignée d'un facteur $\Theta(\log n)$ de l'optimale, facteur non borné donc. C'est deux bonnes raisons pour dire que cette heuristique n'est pas un algorithme d'approximation.

3.1.2 Problème inapproximable

De nombreux problèmes sont non seulement difficiles à résoudre de manière exacte mais restent difficiles à approximer pour tout facteur d'approximation donné. On dit qu'ils sont *inapproximables*. En voici un exemple :

VOYAGEUR DE COMMERCE

Instance: un graphe complet valué (G, ω)
Question: trouver une tournée optimale, c'est-à-dire un cycle passant par tous les sommets une seule fois et de coût minimum.

Ce problème (en fait le problème de décision associé) est connu pour être NP-complet. Mais comme on va le voir, pour tout facteur d'approximation α, aucune α-approximation ne peut exister, sauf si P=NP.

Pour cela on va réduire un problème NP-complet, à savoir HAMILTONISME qui consiste à trouver un cycle passant par tous les sommets d'un graphe une et une seule fois, à celui du VOYAGEUR DE COMMERCE. Soit H une instance de HAMILTONISME ayant n sommets et α un facteur d'approximation quelconque. On a $\alpha \geqslant 1$ car c'est un problème de minimisation. On construit, à partir de H, l'instance (G_H, ω_H) pour le VOYAGEUR DE COMMERCE définit par le graphe $G_H := H \cup \bar{H}$, c'est-à-dire le graphe complet composé des arêtes et non-arêtes de H, et par la valuation ω_H définie par :

$$\omega_H(uv) := \begin{cases} 1 & \text{si } uv \in E(H) \\ 1 + \alpha n & \text{sinon} \end{cases}$$

Soit A un algorithme supposé être une α-approximation du VOYAGEUR DE COMMERCE.

Si $\mathsf{A}(G_H, \omega_H) \leqslant \alpha n$, alors A a trouvé un cycle passant par tous les sommets de G_H une seule fois et n'utilisant que des arêtes de H. Donc H est hamiltonien. En effet, si la tournée calculée par A possède une arête hors de H (dans $\bar{H}$ donc), alors son coût aurait été au minimum de $1 + \alpha n$.

Si $\mathsf{A}(G_H, \omega_H) > \alpha n$, alors H n'est pas hamiltonien. En effet, sinon il existerait une tournée de coût n (celle utilisant un cycle hamiltonien) et donc le facteur d'approximation de A serait $> \alpha$.

Autrement dit, $\mathsf{A}(G_H, \omega_H) \leqslant \alpha n$ si et seulement si H est la hamiltonien. Clairement, sauf si P=NP, A ne peut pas être un algorithme polynomial, et donc ne peut pas être une α-approximation.

En passant, on s'aperçoit que la réduction ne marche pas si l'on s'intéresse à des instances du VOYAGEUR DE COMMERCE où la valuation des arêtes est bornée, disons $\omega(uv) \in [1, K]$ pour une constante K indépendante de n. De fait, il existe de nombreux algorithmes d'approximation pour les instances particulières, où par exemple les valuations représentent une distance (où l'inégalité triangulaire s'applique), voir une distance euclidienne ou encore une généralisation comme les espaces de dimension dou-

blante bornée [3]. Ce sont des instances importantes en pratiques. On parle parfois de *Metric TSP*. On reparlera du Voyageur de commerce au paragraphe 3.4.

Une variante du Voyageur de commerce consiste à trouver dans un graphe valué (pas forcément complet) un cycle (pas forcément simple) de coût minimum. On parle alors de *Graphic TSP*.

Ces variantes sont approximables à un facteur 1.5 pour *Metric TSP* et 1.4 pour *Graphic TSP* [SV14]. Il existe des bornes inférieures sur le meilleur facteur d'approximation possible, si P≠NP, qui sont $1 + 1/122$ pour *Metric TSP*, $1 + 1/534$ pour *Graphic TSP* (cf. [Kar15]). La marge de progrès pour ce problème est donc relativement confortable.

3.1.3 Certificat positif, négatif

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 binaire C qui aide à affirmer que I est une instance positive est un *certificat positif*. On l'appelle aussi *preuve*, *solution*, ou encore *témoin* dans le contexte des calculs probabilistes. 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 $\leqslant k$ présume que C code une solution, soit un ensemble de sommets. Il vérifie si cet ensemble est bien une couverture par sommets pour le graphe et que sa taille est au plus k.

Un certificat négatif pour le problème de Primalité est une factorisation non triviale (autre que par l'unité), ce qui est une preuve de sa non-primalité.

Il faut observer qu'aucune hypothèse n'est faite sur le temps nécessaire pour *trouver* le certificat. C'est le vérificateur qui doit être polynomial. 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 $\mathcal{V}(I, \varepsilon) = \mathsf{A}(I)$ l'algorithme A lui-même, de complexité polynomial, qui est à la fois un vérificateur positif et négatif. Donc, $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$. La question largement ouverte est de savoir s'il n'y a pas égalité.

3. Un espace métrique (V, d) est de *dimension doublante* δ si pour tout point $u \in V$ et tout $r \in \mathbb{R}$ il existe un ensemble S d'au plus 2^δ points tel que $B(u, r) \subseteq \bigcup_{s \in S} B(s, r/2)$. Le plan euclidien est de dimension doublante $\log_2(7)$, un disque de rayon r pouvant toujours être couvert par 7 disques de rayon $r/2$.

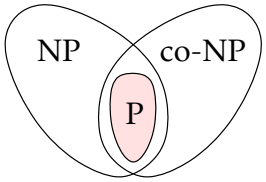


FIGURE 3.2 – P ⊆ NP ∩ co-NP.

3.2 Problèmes bien caractérisés

Les problèmes qui possèdent à la fois un certificat positif et négatif, ceux dans NP ∩ co-NP donc, sont dits « bien caractérisés ». Selon la conjecture NP ≠ co-NP, les problèmes NP-difficiles [4] 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.

3.2.1 Couplage

On rappelle qu'un *couplage* (*matching* en Anglais) est un ensemble indépendant d'arêtes, donc ne partageant aucune extrémité. Un couplage M de G est *maximal* si pour toute arête $uv \in E(G) \setminus M$, $M \cup \{uv\}$ n'est pas un couplage. Il est *maximum* (ou de taille maximum) si tout couplage de G est de taille $\leqslant |M|$ (cf. la figure 3.3). Un couplage qui couvre tous les sommets est dit *parfait*. [*Exercice*. Construire un graphe ayant un couplage parfait et qui ne soit pas biparti.]

FIGURE 3.3 – Couplage maximal de taille 2 et couplage maximum de taille 4 qui est parfait.

La recherche de couplage maximum a de nombreuses applications (comme celles des greffes, cf. la figure 6), mais pas que. Il existe aussi de nombreux théorèmes en théorie des graphes à propos des graphes possédant un couplage parfait. Par exemple, le théorème de Petersen (1891) énonce que tout graphe cubique sans isthme possède un couplage parfait. C'est l'un des plus anciens théorèmes de la théorie des graphes. De manière générale, le théorème de Tutte énonce qu'un graphe G possède un couplage parfait si et seulement si, pour tout sous-ensemble U, $G \setminus U$ possède au plus $|U|$ composantes connexes avec un nombre impair de sommets. Et la formule de Tutte-Berge

4. Un problème est dit NP-difficile s'il est au moins aussi difficile que n'importe quel problème NP (réduction), mais, contrairement aux problèmes NP-complets, il n'est pas forcément dans NP.

donne la taille du couplage maximum de G qui vaut

$$\frac{1}{2} \min_{U \subseteq V(G)} \{|U| - \mathrm{nci}(G \setminus U) + |V(G)|\}$$

où $\mathrm{nci}(H)$ est le nombre de composantes connexes du graphe H ayant un nombre impair de sommets. Cette formule, malheureusement, ne donne pas de moyen efficace pour calculer cette taille.

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

Algorithme CouplageMaximal(G)

1. $M := \varnothing$
2. Tant qu'il existe $uv \in E(G)$, faire :
   (a) $M := M \cup \{uv\}$
   (b) $G := (G \setminus \{u\}) \setminus \{v\}$

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 complexe. Le premier algorithme polynomial date de 1965 dû à Jack Edmonds [Edm65] et de complexité $O(n^3)$. Cependant, c'est plus simple dans les graphes bipartis.

On rappelle qu'un graphe G est *biparti* si l'ensemble de ses 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.

Calcul de couplage dans les bipartis. Dans un graphe biparti à n sommets et m arêtes, on peut calculer un couplage maximum en temps $O(m\sqrt{n}) = O(n^{5/2})$ par l'algorithme d'Hopcroft et Karp [HK73]. Il existe même une implémentation qui prend un temps $O(n^{3/2}\sqrt{m/\log n}) = O(n^{5/2}/\sqrt{\log n})$ [ABMP91] ce qui est meilleur si $m > n^2/\log n$ (cas de graphes très denses). Plus récemment encore [MS04] a trouvé un algorithme en $O(n^\omega)$ où [8] $\omega < 2.3729$, mieux que Hopcroft-Karp si $m > n^{\omega-1/2} > n^{1.87}$.

Calcul de couplage via un flot maximum. Un couplage dans un biparti peut aussi être calculé en utilisant une technique de flot maximum dans un graphe orienté (cf. la définition précise page 10). 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. Voir la figure 3.4. S'il existe un flot de valeur $k \in \mathbb{N}$, alors G possède un couplage de taille k. Et, inversement, si G possède un couplage de taille k, il existe un flot de valeur $k \in \mathbb{N}$. L'algorithme de Ford-Fulkerson [FF56] produit assez simplement une solution en temps $O(nm) = O(n^3)$.

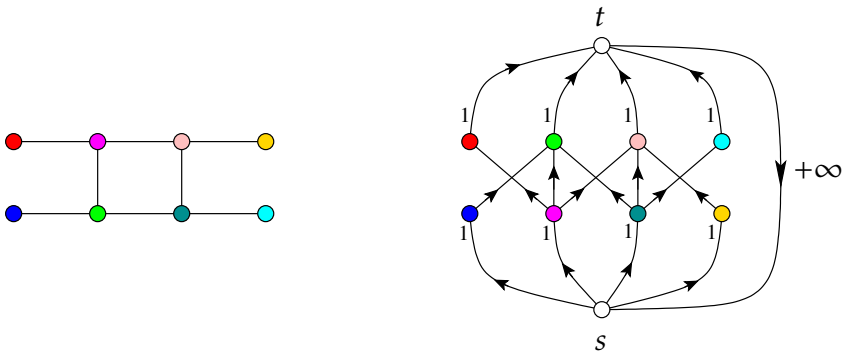


FIGURE 3.4 – Réduction à un problème de flot maximum pour calculer un couplage maximum d'un graphe biparti.

Calcul de flot maximum avec capacités unitaires. Dans le cas de capacité unitaire, Even et Tarjan [ET75] fait mieux avec un algorithme en $O\left(\min\left\{\sqrt{m}, n^{2/3}\right\} m\right)$ découvert indépendamment par Karzanov [Kar73]. Le record actuel est détenu par l'algorithme de Madry [Mąd13] qui est de complexité $m^{10/7} \log^{O(1)} n$, ce qui est toujours mieux que Ford-Fulkerson. [*Question.* Pourquoi ?] [Mąd13] montre aussi une réduction du problème de flot maximum à capacité unitaire à celui du b-couplage maximum, une généralisation du couplage maximum. Dans cette variante, chaque sommet u a une demande $b(u) \geqslant 0$, et il s'agit de trouver un multi-ensemble d'arêtes de cardinalité [5] maximum, une même arête pouvant apparaître plusieurs fois, sans avoir deux arêtes différentes incidentes au même sommet (couplage) et sans avoir plus de $b(u)$ copies d'arêtes incidentes à u.

Calcul flot maximum avec capacités arbitraires. Lorsque les capacités ne sont pas unitaires, mais entières [6] et bornées par C, alors Ford-Fulkerson est en $O(nmC)$. Il existe un algorithme en $O\left(\min\left\{\sqrt{m}, n^{2/3}\right\} m \log\left(n^2/m\right) \log C\right)$, dû à Goldberg-Rao [GR98]. L'algorithme de Madry quant à lui réalise $(mC)^{10/7} \log^{O(1)} n$. Il existe des algorithmes indépendants de C, en $O(nm)$ obtenus en combinant les algorithmes de King-Rao-Tarjan [KRT94] et d'Orlin [Orl13].

Les algorithmes de couplage dans les graphes arbitraires sont généralement basé sur l'idée de trouver un chemin *alternant* entre deux sommets non couverts par le couplage courant M, c'est-à-dire un chemin alternant des arêtes de M et de $E(G) \setminus M$. Une fois un tel chemin trouvé, on inverse les arêtes du chemin qui font partie de M avec celles qui n'en font pas partie. On augmente alors le couplage d'une arête (en fait on couvre deux sommets de plus) car les sommets de départ et d'arrivée deviennent couverts alors qu'ils ne l'étaient pas dans M. Les contraintes des sommets internes ne sont pas modifiées car

5. La cardinalité d'un multi-ensemble est le nombre total d'éléments avec multiplicités.
6. Il est bien connu que si les capacités sont irrationnelles, les algorithmes comme celui de Ford-Fulkerson peuvent ne pas terminer [Zwi95].

tous ces sommets étaient déjà couverts par M.

Plus formellement (voir aussi la figure 3.5) :

Algorithme CouplageMaximum(G)

1. $M := \varnothing$
2. Tant qu'il existe un chemin P alternant entre deux sommets non couverts par M faire $M := (M \setminus E(P)) \cup (E(P) \setminus M)$.
3. Renvoyer M

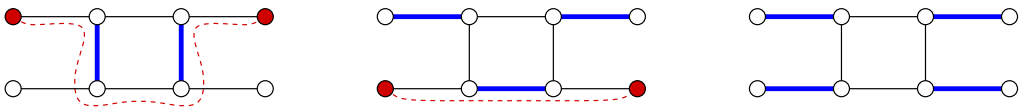


FIGURE 3.5 – Application de l'algorithme CouplageMaximum à partir d'un couplage maximal.

Cette méthode incrémentale converge vers un couplage maximum car une propriété due à Berge stipule :

Propriété 3.1 (Berge, 1957) *Un couplage est maximum si et seulement s'il n'existe pas de chemin alternant entre deux sommets non couverts.*

Preuve. Un sens est facile : si G possède un chemin alternant entre deux sommets non couverts par M, c'est que M n'est pas maximum. Montrons le contraire. Supposons que M n'est pas maximum. Il reste à montrer que G possède un chemin alternant entre deux sommets non couverts par M.

Soit M' un couplage tel que $|M'| > |M|$. On considère le sous-graphe G' de G induit par les arêtes de la différence symétrique entre M et M', c'est-à-dire contenant uniquement les arêtes de $M' \setminus M$ et de $M \setminus M'$, voir figure 3.6. Il s'agit d'un graphe de degré

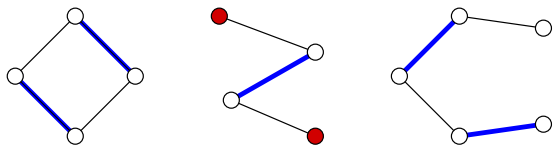


FIGURE 3.6 – Graphe (non connexe) induit par la différence symétrique de deux couplages, M (en bleu) et M' (en noir) avec $|M'| > |M|$, et contenant un chemin alternant.

maximum deux, puisqu'un sommet est au plus incident à une arête de M' et de M. Ses composantes connexes sont donc composées de chemins et de cycles où les arêtes de M et M' alternent. L'alternance implique que les cycles sont tous de longueurs paires. Comme $|M'| > |M|$, il doit donc exister dans G' au moins une composante connexe P qui est un chemin ayant plus d'arêtes de M' que de M. Il doit donc commencer et finir par

une arête de M'. Dans G, P est le chemin alternant recherché. □

La question qui se pose est la mise en œuvre de cet algorithme pour le rendre efficace. L'algorithme d'Edmonds [Edm65], est basé sur le même principe mais inclut la contraction de *blossom* (cycle alternant de longueur impaire). D'autres algorithmes, dont ceux d'Even et Kariv en 1975, de Bartnik en 1978, ont une complexité un peu meilleure en $O\left(\min\left\{n^{5/2}, m\sqrt{n}\log n\right\}\right)$. Tous ces algorithmes sont réputés pour être très complexes.

3.2.2 Les problèmes min-max

Considérons les deux problèmes de décisions suivants :

(Π_1) Couverture par sommets de taille $\leqslant k$?
(Π_2) Couplage de taille $\geqslant \ell$?

Ces deux problèmes sont dans NP car ils ont clairement des certificats positifs. Ont-ils des certificats négatifs ? En reliant ce problème de minimisation à l'autre qui de maximisation, alors on va pouvoir construire des certificats négatifs.

Théorème 3.1 (König-Egavary, 1931) *Pour tout graphe biparti G,*

$$\max_{M\in\mathcal{M}} |M| \;=\; \min_{S\in\mathcal{C}} |S|$$

où $\mathcal{M}$ et $\mathcal{C}$ représentent respectivement l'ensemble des couplages et des couvertures par sommets de G.

Ces deux problèmes ont alors des certificats négatifs si G est biparti. En effet, si la réponse à (Π_1) est NON, alors la couverture minimum est $> k$ et donc il doit exister un couplage de taille $> k$. C'est un certificat positif pour (Π_2), qui est donc un certificat négatif pour (Π_1). Inversement, si la réponse à (Π_2) est NON, alors le couplage maximum est $< \ell$ et donc il doit exister une couverture par sommets de taille $< \ell$. C'est un certificat positif pour (Π_1), qui est donc un certificat négatif pour (Π_2). Donc les restrictions de (Π_1) et (Π_2) aux graphes bipartis sont dans NP ∩ co-NP.

De telles relations min-max permettent de prouver que des problèmes sont bien caractérisés est c'est un premier pas vers la découverte d'algorithmes polynomiaux. En fait, (Π_1) et (Π_2) restreint aux graphes bipartis sont effectivement dans P.

La preuve du théorème 3.1 est constructive. Elle explique en particulier comment construire efficacement une couverture par sommets minimale S à partir d'un couplage maximum M, que l'on peut calculer avec un algorithme de flot maximum (voir page 133).

Preuve du théorème 3.1. Pour tout couplage M et couverture par sommets S de G, il est clair que $|S| \geqslant |M|$ puisqu'il faut au moins un sommet de S pour couvrir chaque arête

de M qui sont indépendantes. En particulier, $\min|S| \geqslant \max|M|$. Il reste donc à montrer qu'à partir d'un couplage maximum M on peut construire une couverture par sommets S de taille $|S| \leqslant |M|$.

Soient A, B la partition des sommets du graphe biparti G. Soit Z_0 l'ensemble des sommets de A qui ne sont pas couverts par M. On forme ensuite l'ensemble Z à partir de Z_0 en ajoutant tous les sommets atteignables à partir de Z_0 par des chemins alternant une arête pas dans M avec une arête de M. Notons que Z_0 et Z peuvent être vides si M est parfait. On définit $S = (A \setminus Z) \cup (B \cap Z)$. Voir la figure 3.7 pour un exemple de construction.

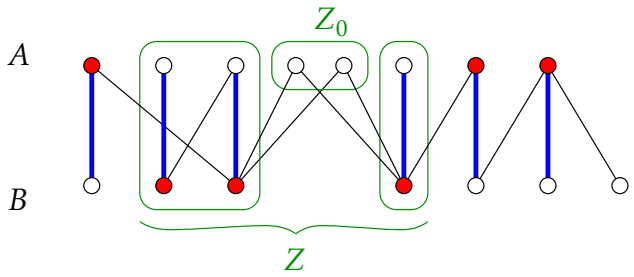


FIGURE 3.7 – Construction d'une couverture par sommets de taille minimale (sommets rouge) à partir d'un couplage maximum dans un graphe biparti. Les sommets de Z, encadrés de vert, peuvent être construit à l'aide d'arbres alternants.

On va d'abord montrer que $|S| \leqslant |M|$. Pour cela il suffit de montrer que (1) si $uv \in M$, alors au plus une des deux extrémités est dans S, c'est-à-dire u et v ne sont pas tous les deux dans S ; et (2) chaque sommet $s \in S$ est couvert par M.

Pour le point (2), supposons $s \in A$. Alors $s \notin Z$ et donc $s \notin Z_0$. Donc s est couvert puisque par définition Z_0 contient tous les sommets de A qui ne sont pas couverts. Supposons $s \in B$. Alors $s \in Z$. Il existe donc un chemin alternant P de Z_0 à s. L'arête ws précédant s dans P n'est pas dans M car P alterne dans l'ordre des arêtes pas dans M et dans M. Donc on arrive toujours dans un sommet de A après une arête de M et un sommet de B après une arête pas dans M. Si s n'est pas couvert, le sous-chemin de P allant jusqu'à s est alternant avec ses extrémités non couvertes. Par la propriété 3.1 de Berge, M n'est pas maximum : contradiction. Donc s est couvert. Il suit que tous les sommets de S sont couverts par M, ce qui démontre le point (2).

Montrons le point (1). Soit $uv \in M$ avec $u \in A$ et $v \in B$. Si u et v sont dans S, c'est que $u \notin Z$ et $v \in Z$. Il existe donc un chemin alternant de Z_0 à v. Comme on vient de le voir, l'arête précédente $wv \notin M$ et comme $w \in Z$, on a $w \neq u$. On a donc un chemin alternant comprenant les arêtes wv et vu ce qui par maximalité de Z implique que $u \in Z$: contradiction. Soit u soit v n'est pas dans S. On a prouvé (1) & (2), donc $|S| \leqslant |M|$.

Il reste à montrer que S couvre toutes les arêtes du graphe. Soit uv une arête de G, avec $u \in A$ et $v \in B$. Supposons $u \in Z$ et $v \notin Z$, puisque sinon u ou v est dans S. Si

$u \in Z_0$, alors $v \in Z$ puisque l'arête uv forme un chemin alternant : contradiction. Si $u \in Z \setminus Z_0$, alors u est couvert par M. L'arête précédente wu du chemin alternant P de u doit être dans M car $u \in A$. Il suit que l'arête $uv \notin M$ sinon M ne serait pas un couplage. On peut donc étendre P à l'arête uv ce qui par maximalité de Z implique que $v \in Z$: contradiction. Donc soit u soit v est dans S, ce qui démontre que S est une couverture par sommets. □

Notons que dans le cas d'un graphe général, il n'y a pas forcément égalité entre le couplage maximum et couverture minimum. D'ailleurs, le problème de couplage est polynomial alors qu'il est NP-complet pour la couverture par sommets. [*Exercice*. Donner un exemple de graphe avec un couplage maximum de taille k et de couverture par sommets minimum de taille $2k$.]

[*Exercice*. Considérons les problèmes EDC : Ensemble Dominant Connexe de taille $\leqslant k$; et ACF : Arbre Couvrant avec $\geqslant \ell$ Feuilles. Sachant que, pout tout graphe connexe G, $\mathrm{EDC}_{\min}(G) = |V(G)| - \mathrm{ACF}_{\min}(G)$, peut-on dire que les problèmes EDC et ACF sont bien caractérisés ?]

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, ainsi qu'un algorithme capable de se mesurer à ce minorant. (Voir la figure 3.8). Pour un problème de maximisation, c'est le contraire : il faut trouver un majorant le plus petit possible.

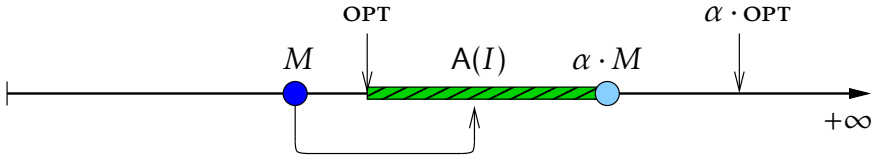


Figure 3.8 – L'algorithme A, ici de minimisation, se base sur une borne inférieure M de opt calculable en temps polynomiale. S'il est correct, pour toute instance I, $\mathrm{opt} \leqslant \mathsf{A}(I)$ nécessairement. Il faut montrer que $\mathsf{A}(I) \leqslant \alpha \cdot M$ pour établir que $\mathsf{A}(I) \leqslant \alpha \cdot \mathrm{opt}$ et qu'il s'agit d'une α-approximation. Pour un problème de maximisation, il faut inverser la flèche de l'axe.

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 d'abord trouver un bon majorant pour OPT et le relier à notre construction. Pour le majorant, on a OPT $\leqslant m$. Maintenant, l'un des deux ensembles définis précédemment est de taille $\geqslant m/2$. C'est donc une 1/2-approximation.

Voici un autre exemple assez proche qui va illustrer la méthode dite « probabiliste » (cf. [AS00]). Cette méthode permet de montrer que certains objects combinatoires, typiquement un graphe ayant une propriété donnée, existent bel et bien sans avoir à les construire explicitement. Pour ce faire on élabore un algorithme probabiliste que l'on analyse. Si on arrive à montrer que l'algorithme a une probabilité non nulle de fournir un des objets recherchés, c'est que l'objet existe.

SOUS-GRAPHE BIPARTI MAXIMUM

Instance: un graphe G
Question: trouver un sous-graphe biparti de G avec un nombre maximum d'arêtes.

Une solution probabiliste consiste à colorier les sommets avec deux couleurs, disons blanc ou noir, chacune avec probabilité 1/2. Les arêtes du sous-graphe solution H sont les arêtes dont les extrémités ont des couleurs différentes. On vérifie aisément que H est biparti.

Soit $X_e \in \{0, 1\}$ la variable aléatoire indiquant si l'arête e de G apparaît aussi dans H (=1) ou pas (=0). La probabilité qu'une arête e soit dans H est $\Pr(X_e = 1) = 2 \cdot 1/4 = 1/2$, ce qui correspond aux deux événements parmi les quatre paires de couleurs possibles : noir-blanc ou blanc-noir. L'espérance de X_e vaut $E(X_e) = 0 \cdot \Pr(X_e = 0) + 1 \cdot \Pr(X_e = 1) = 1/2$. La moyenne du nombre d'arêtes de H est $E(\sum_e X_e) = \sum_e E(X_e) = m/2$ où m est le nombre d'arêtes de G.

L'algorithme ainsi défini n'est pas formellement une 1/2-approximation, car même s'il est vrai que le nombre d'arêtes de la solution optimale vérifie $m/2 \leqslant \text{OPT} \leqslant m$, la solution construite par l'algorithme contient $m/2$ arêtes seulement en moyenne. [*Exercice.* Quelle est la probabilité d'avoir au moins $m/2$ arêtes ?] Notons que cette méthode permet déjà d'affirmer que tout graphe possède un sous-graphe biparti contenant au moins la moitié des arêtes initiales. [*Question.* Pourquoi ?]

Et *quid* de l'algorithme glouton naïf : on considère chaque arête dans un ordre quelconque et on l'ajoute à la solution s'il elle ne crée pas de cycle impair, soit l'algorithme de Kruskal auquel on ajoute la contrainte « impair ». Le résultat est bien un graphe biparti par construction. Mais c'est quand même une mauvaise idée. Si le graphe G est une clique et qu'on ajoute en premier toutes les arêtes incidentes à un sommet donné, alors on ne pourra mettre que $n - 1$ arêtes puisqu'ensuite tout autre arête crée un triangle. Cependant on aurait pu construire un biparti complet et mettre $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \approx n^2/4$ en mettant toutes les arêtes possibles entre deux ensembles disjoints de $n/2$ sommets. Cette méthode garantie $n - 1$ arêtes seulement pour un graphe connexe, soit le nombre d'arêtes d'un arbre couvrant.

En fait, on peut construire de manière déterministe et en temps linéaire un sous-graphe biparti avec au moins $m/2$ arêtes. Notons A et B la partition recherchée des sommets. Au départ $A = B = \varnothing$. Puis, pour chaque sommet u (listé dans n'importe quel ordre) on fait : si u a plus de voisin dans A que dans B, alors $B := B \cup \{u\}$ sinon $A := A \cup \{u\}$. Le sous-graphe est composé de toutes les arêtes reliant A à B. L'analyse est la suivante. Si on note $F(u)$ l'ensemble des arêtes reliant u aux sommets de $A \cup B$ au moment où u est considéré. Alors, $\{F(u)\}_u$ forme une partition des arêtes de G car pour qu'une arête $uv \in F(u)$ soit de nouveau considérée par l'algorithme il faut soit u soit $v \notin A \cup B$ ce qui n'est plus le cas après la visite de u. Donc $\sum_{u \in V(G)} |F(u)| = m$. La règle garantit pour chaque sommet u qu'au moins $\frac{1}{2}|F(u)|$ arêtes de G sont ajoutées au sous-graphe biparti final, soit un total d'au moins $\sum_u \frac{1}{2}|F(u)| = m/2$.

[*Exercice*. Trouver une instance critique. Est-ce que K_n est une instance critique, sachant que $K_{n/2,n/2}$ a deux fois moins d'arêtes ? Est-ce que $K_{n/2,n/2}$ est une instance critique ?]

3.3 Couverture par sommets de taille minimum

Dans cette partie on cherche à déterminer une couverture par sommets de cardinalité minimum. On rappelle que le problème de décision associé est NP-complet.

3.3.1 Un premier algorithme

Algorithme Approx_VC1(G)

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

Proposition 3.1 *Approx_VC1 est une 2-approximation pour Couverture par sommets de taille minimum.*

Preuve. Il faut montrer trois choses :

1. L'algorithme est polynomial. Ce point est évident, l'algorithme CouplageMaximal est polynomial.
2. L'algorithme renvoie une couverture par sommets. Supposons qu'il existe une arête uv du graphe non couverte par S. Comme $u \notin S$, aucune arête incidente à u n'est dans M. De même, $v \notin S$ implique qu'aucune arête incidente à v n'est dans M. Il suit que $uv \notin M$ et que M n'est pas maximal : contradiction.

3. La couverture renvoyée est de taille $\leqslant 2 \cdot \text{OPT}$. Il faut au moins un sommet par arête d'un couplage, quel que soit le couplage. En particulier, toute couverture doit avoir une taille $|M|$. Donc $|M| \leqslant \text{OPT}$. Or la couverture S retournée vérifie $|S| \leqslant 2|M|$. Donc $|S| \leqslant 2|M| \leqslant 2 \cdot \text{OPT}$.

□

[*Exercice*. Donner une instance critique.]

3.3.2 Un second algorithme

Le prochain algorithme est un peu plus facile à programmer pour 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 donc enlever tous les sommets isolés et appliquer l'algorithme à chacune des composantes connexes, ce qui est facile à traiter.

Algorithme Approx_VC2(G)

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

Proposition 3.2 *Approx_VC2 est une 2-approximation pour* Couverture par sommets *de taille minimum.*

Preuve. Comme précédemment, on a trois choses à montrer :

1. L'algorithme est polynomial, il est clairement linéaire.

2. $S = \text{Approx\_VC2}(G)$ est une couverture pour G. En effet, sinon, il existe une arête uv 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. On note donc en passant que les feuilles d'un arbre couvrant calculé par un DFS forme toujours un ensemble indépendant du graphe.

3. Montrons que $\text{OPT} \geqslant |S|/2$. Pour cela, on va construire un couplage M pour G de taille $\geqslant |S|/2$ (cf. figure 3.9). 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 des sommets de S à distance dans T modulo 2 égale à i depuis la racine de T. On choisit $W \in \{S_0, S_1\}$ comme l'ensemble des sommets de S coloriés de la même couleurs et dont la couleur est la plus représentée. Clairement, $|W| \geqslant |S|/2$. On associe pour chaque sommet $u \in W$ une arête, notée $e_u = uv$ 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. Soit $e_u = uv$ et $e_{u'} = u'v'$ deux arêtes distinctes de M. Si $u = u'$, alors v et v' aurait le même père ce qui est exclu par l'unicité du choix du fils. Donc $u \neq u'$. Le cas $v = v'$ est également exclu car T est un arbre. Donc $v \neq v'$. Reste le cas $u' = v$ (ou de manière symétrique $u = v'$). Si $u' = v$, alors u et u' sont adjacents dans T ce qui est interdit par la 2-coloration. Donc les arêtes e_u et $e_{u'}$ sont indépendantes. Conclusion M est un couplage de T donc de G de taille $|M| = |W| \geqslant |S|/2$. Précédemment on a vu que $\text{OPT} \geqslant |M|$ donc $|S| \leqslant 2|M| \leqslant 2 \cdot \text{OPT}$. C'est une 2-approximation.

□

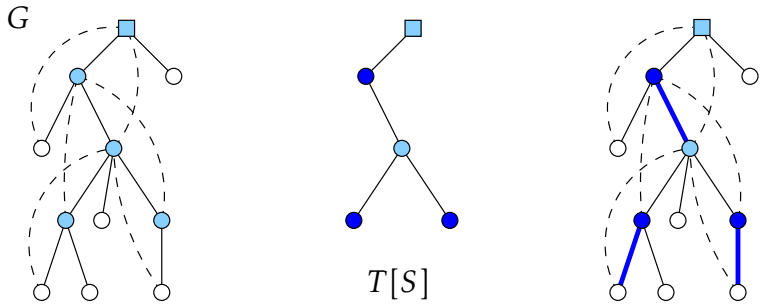


FIGURE 3.9 – Construction d'une couverture par sommets S (en bleu clair) à l'aide des sommets internes d'un arbre en profondeur T (arêtes pleines) ainsi que la preuve que le graphe G contient un couplage de taille au moins $|S|/2$ à l'aide d'une 2-coloration de $T[S]$.

[*Exercice*. En considérant les sommets de S adjacents à une feuille de T, montrez que l'arbre de la figure 3.9 possède un couplage de taille au moins 5. Est-ce que, de manière générale, ce minorant (le nombre de sommets de S qui sont adjacents à au moins une feuille de T), est un « bon » minorant pour l'agorithme Approx_VC2, c'est-à-dire un minorant permettant de borner le facteur d'approximation ?]

[*Exercice*. Est-ce que algorithme Approx_VC2 est toujours correct si au lieu de renvoyer les sommets internes de T on renvoie tous les sommets de T de degré > 1 ?]

[*Exercice*. Donnez une instance critique pour Approx_VC2 (et pour un graphe connexe).]

3.3.3 Technique de pré-calcul (*color coding*)

Théorème 3.2 *Si G peut être proprement 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 3.2 implique un algorithme exact avec $k = 2$.

Pour cela on aura besoin du lemme suivant, dont la preuve constructive est similaire à celle vue dans le lemme 2.3 du chapitre 2.

Lemme 3.1 *En temps $O(m\sqrt{n})$ on peut partitionner les sommets d'un graphe 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 est de taille au moins $|P| + \frac{1}{2}|Q|$.*

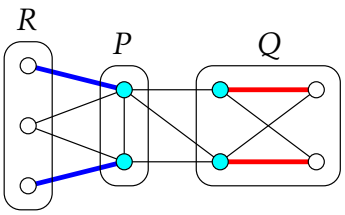


FIGURE 3.10 – Partition P, Q, R du lemme 3.1 construit à partir d'une couronne R.

Preuve. On construit de manière identique au lemme 2.3 une couronne R de G (voir définition 2.4), et l'on pose $P = N(R)$ et $Q = V(G) \setminus (R \cup P)$. Plus précisément, on commence par calculer un couplage maximal (en temps linéaire), arêtes que l'on enlève de G pour obtenir un graphe biparti dans lequel on calcule un couplage maximum (cette fois-ci en temps $O(m\sqrt{n})$, voir moins, cf. le paragraphe 3.2.1). La façon dont est définie la couronne R (et son voisinage P) fait que le couplage maximum dans le graphe restant ($G[Q]$) est parfait [*Cyril.* Est-ce si clair ?] : il couvre tous les sommets de Q.

Il découle de la construction que : (1) toute couverture par sommets pour $G[Q]$ (et donc pour G) doit contenir au moins $|Q|/2$ sommets ; et (2) toute couverture optimale contient P. Comme les sommets de P ne peuvent pas couvrir les arêtes de $G[Q]$ (mais seulement celles entre P et Q), la taille de toute couverture doit être de taille au moins $|P| + |Q|/2$. Voir la figure 3.10. □

Preuve du théorème 3.2. On calcule les ensembles P, Q, R du lemme 3.1 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. C'est la couverture qui va être renvoyée par l'algorithme.

Dans un premier temps, on choisit $C \subseteq Q$ comme une couverture par sommets quelconque de $G[Q]$. Et soit $\alpha = |C|/|Q|$. Évidemment C doit contenir au moins la moitié des sommets de Q puisque sinon $P \cup C$ serait une couverture de G de taille $|P \cup C| = |P| + |C| < |P| + \frac{1}{2}|Q|$ contredisant la propriété 3 du lemme 3.1. Il suit que $\alpha \geqslant 1/2$.

Montrons que la couverture $P \cup C$ pour G vérifie $|P \cup C| \leqslant 2\alpha \cdot \textsc{opt}(G)$. En effet, d'après la propriété 3 du lemme 3.1, et en utilisant le fait que $\frac{1}{2\alpha} \leqslant 1$:

$$\textsc{opt}(G) \geqslant |P| + \frac{1}{2}|Q| = |P| + \frac{1}{2\alpha}|C| \geqslant \frac{1}{2\alpha}(|P| + |C|) = \frac{1}{2\alpha}|P \cup C| . \tag{3.1}$$

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

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 $|I| \geqslant |Q|/k$ 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 $|C| \leqslant (1 - 1/k) \cdot |Q|$, et en particulier $\alpha = |C|/|Q| \leqslant 1 - 1/k$.

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

$$|P \cup C| \leqslant 2\alpha \cdot \textsc{opt}(G) \leqslant (2 - 2/k) \cdot \textsc{opt}(G) .$$

□

Nous donnons maintenant quelques applications du théorème 3.2. Ci-dessous la « dégénérescence » de quelques familles graphes, voir la définition 2.2.

- graphes sans arêtes 0-dégénérés
- arbres ou forêts 1-dégénérés
- graphes planaires-extérieurs et grilles 2D 2-dégénérés
- graphes cubiques 3-dégénérés
- graphes planaires 5-dégénérés
- grilles à d-dimensions ..... d-dégénérés
- graphes de degré maximum d ..... d-dégénérés
- graphes de largeur arborescente au plus k ..... k-dégénérés
- graphes qui excluent un mineur à k sommets ..... $O(k\sqrt{\log k})$-dégénérés

Notons que ces dégénérescences résultent à peu près toute du fait que toute famille de graphes *héréditaire* à n sommets et au plus $f(n)$ arêtes est $\lfloor 2f(n)/n \rfloor$-dégénérée. [*Exercice.* Pourquoi ?] On rappelle qu'une famille de graphes (ou une propriété de graphes) est *héréditaires* si elle est stable par sous-graphes induits. C'est bien sûr le cas des graphes 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 3.11 pour un exemple). La complexité de cet algorithme est $O(n+m) = O(kn)$. [*Question*. Pourquoi ? Comment ?] Donc, pour les graphes k-dégénérés il existe une $(2-2/(k+1))$-approximation pour Couverture par sommets calculable en temps $O(kn^{3/2})$.

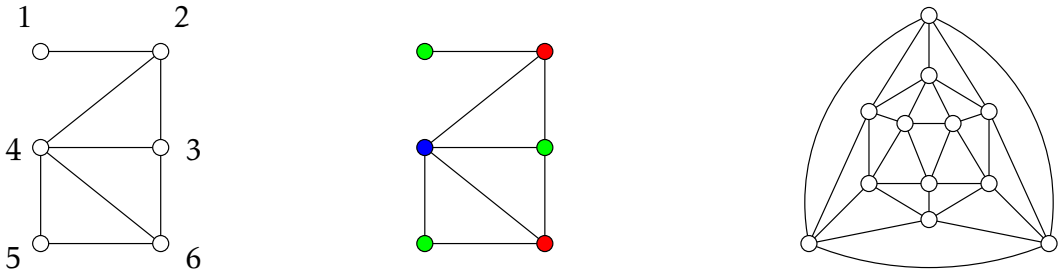


Figure 3.11 – Coloration d'un graphe planaire par FirstFit. Le numéro des sommets est l'ordre d'épluchage, l'ordre des couleurs étant rouge-vert-bleu. [*Question*. Combien de couleurs produit FirstFit pour l'icosaèdre dessiné à droite ?]

Pour les graphes planaire, il est possible d'obtenir une 4-coloration en temps polynomial. (Attention ! l'algorithme est non trivial !) Donc, grâce au théorème 3.2, on obtient une 3/2-approximation pour Couverture par sommets dans les graphes planaires. Cependant il existe un algorithme plus simple et linéaire donnant une 5/3-approximation, mais aussi une $(1+\Theta(1/\sqrt{\log\log n}))$-approximation pour un temps de calcul en $O(n\log n)$, voir [BYE82][HC06].

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

[*Exercice*. Soit $V_1 \cup \cdots \cup V_k$ une partition des sommets de G calculable en temps polynomial telle que $\chi(G[V_i])$ est calculable en temps polynomial. Montrez qu'il existe une k-approximation pour $\chi(G)$.]

3.3.4 Technique par programmation linéaire (LP)

Le problème de Couverture par sommets de taille minimum peut facilement être résolu par la technique de programmation linéaire (LP pour *Linear Programming*). Et de nombreux solveurs très efficaces existent. Les solveurs Sat discutés page 44 sont plus adaptés aux problèmes de décisions, le codage de la contrainte « ... de taille k » par exemple étant beaucoup moins naturel.

Dans la formulation ici choisie, chaque sommet u du graphe G reçoit une variable $x_u \in \{0,1\}$ telle que $x_u = 1$ si u est choisi pour être dans la couverture, et $x_u = 0$ sinon. On souhaite minimiser la taille de la couverture, soit le nombre de variables à 1. Cela

revient à minimiser la somme de toutes les variables, avec bien sûr la contrainte que si $uv \in E(G)$, alors x_u ou x_v vaut 1, ce qui peut s'écrire dans ce cas $x_u + x_v \geqslant 1$.

Pour résumer :

$$\begin{aligned} & \min \sum_{u \in V(G)} x_u & & \text{(S1)} \\ \text{tel que :} \quad & x_u + x_v \geqslant 1 & \forall uv \in E(G) & \\ & x_u \in \{0,1\} & \forall u \in V(G) & \end{aligned}$$

Malheureusement, la programmation linéaire en nombre entier est NP-difficile, ce qui n'est pas choquant étant donné que le problème Couverture par sommets l'est aussi. Cependant, la programmation linéaire en nombre fractionnaire est polynomiale. On va donc : (1) relaxer les contraintes entières, dire que $x_u \in [0,1]$ au lieu de $x_u \in \{0,1\}$ ce qu'on peut même simplifier en $x_u \geqslant 0$; [*Question*. Pourquoi peut-on se passer de la condition $x_u \leqslant 1$?] (2) trouver une solution fractionnaire ; puis (3) en déduire une solution entière valide mais pas forcément optimale. C'est la méthode dite de « relaxation ».

Le système (S1) devient alors (seule la dernière ligne change) :

$$\begin{aligned} & \min \sum_{u \in V(G)} x_u & & \text{(S2)} \\ \text{tel que :} \quad & x_u + x_v \geqslant 1 & \forall uv \in E(G) & \\ & x_u \geqslant 0 & \forall u \in V(G) & \end{aligned}$$

Avant de présenter l'algorithme à proprement parlé, notons que l'espace de recherche défini par le système (S2) est plus grand que celui défini par le système (S1). Il le contient même strictement : $x_u \in \{0,1\} \subset \mathbb{R}^+$. Donc une solution pour (S2), ici un minimum, sera plus petite ou égale à celle de (S1). Pour prendre un exemple un peu moins abstrait, à l'Université de Bordeaux, la taille minimum d'un étudiant en Informatique est plus grande ou égale à la taille minimum d'un étudiant en Sciences & Technologies.

Ainsi, la solution de (S2), qui est calculable en temps polynomial, donne une borne inférieure à la solution exacte donnée par (S1), qui elle n'est pas calculable polynomialement sauf si P=NP. On va se servir de cette borne inférieure pour en déduire un algorithme d'approximation.

L'algorithme est le suivant :

Algorithme Approx_VC3(G)

1. Chercher une solution x^* du système (S2) pour G.
2. Renvoyer l'ensemble $S := \left\{u : x^*_u \geqslant \frac{1}{2}\right\}$.

Proposition 3.3 *Approx_VC3 est une 2-approximation pour Couverture par sommets de taille minimum.*

Preuve. Comme précédemment, on a trois choses à montrer :

1. L'algorithme est polynomial, chacune des deux étapes étant polynomiale.
2. Montrons que $S = \text{Approx\_VC}_3(G)$ est une couverture par sommets de G. En effet, comme x^* est une solution du système (S2), alors pour toute arête $uv \in E(G)$ on a $x^*_u + x^*_v \geqslant 1$. En particulier, soit $x^*_u \geqslant \frac{1}{2}$, soit $x^*_v \geqslant \frac{1}{2}$. Par conséquent, soit $u \in S$, soit $v \in S$.
3. Montrons que $\text{OPT}(G) \geqslant |S|/2$. On a déjà vu que $\text{OPT}(G) \geqslant \sum_{u \in V(G)} x^*_u$, car $\text{OPT}(G)$ est la solution du système (S1) dont l'espace de recherche est inclus dans celui de (S2). Donc :
$$\text{OPT}(G) \geqslant \sum_{u \in V(G)} x^*_u \geqslant \sum_{u \in S} x^*_u \geqslant \sum_{u \in S} \frac{1}{2} \geqslant \frac{1}{2}|S| \,.$$

□

[*Exercice*. Construisez une instance critique pour $\text{Approx\_VC}_3(G)$.]

Le *gap d'intégralité* (*integrality gap* en Anglais) est le rapport maximum entre la solution optimale, issue de la formulation exacte, et la solution issue de la version relaxée. Plus précisément, pour un problème de minimisation [7] Π et une formulation F en programmation linéaire factionnaire donnés, le gap d'intégralité est le ratio suivant :

$$\text{gap}_{\Pi,F} := \sup_{I \in \Pi} \frac{\text{OPT}_\Pi(I)}{\text{OPT}_F(I)} \,.$$

Notons que différentes formulations peuvent mener à différents gaps. La proposition 3.3 montre que le gap pour Couverture par sommets (et cette version fractionnaire) n'est pas plus que 2. La question est : Peut-on faire mieux ? C'est-à-dire moins que 2 ?

Considérons une clique K_n. Il est clair que $\text{OPT}(K_n) = n - 1$ car si deux sommets u, v ne sont pas dans la solution, l'arête uv ne sera pas couverte. Cependant, $x^*_u = \frac{1}{2}$ est une solution du programme linéaire fractionnaire dont la somme totale est seulement de $n/2$. Du coup, le gap est ici $(n-1)/(n/2) = 2 - 2/n$ ce qui tend vers 2 quand n devient grand. [*Exercice*. Pourquoi une instance critique pour Approx_VC_3 ne sert a priori à rien pour montrer que le gap d'intégralité, pour la formulation proposée, est > 1 ?]

Comme dit précédemment, le meilleur facteur d'approximation actuel est $2 - \Theta(1/\sqrt{\log n})$, soit légèrement mieux que ce qui peut être réalisé par la programmation linéaire.

Et pour les graphes planaires ? On peut montrer que le gap est au plus 3/2 (voir l'algorithme ci-après), et la borne est asymptotiquement atteinte à cause de K_4 (voir ci-dessus). Cependant, comme déjà mentionné page 145, il existe des algorithmes en $O(n)$ donnant des 5/3-approximations.

7. Pour un problème de maximisation, la ratio est inverse.

Algorithme PlanarApprox_VC(G)

1. Chercher une solution x^* du système (S2) pour G.
2. Calculer un ensemble indépendant I de taille au moins $n/4$.
3. Renvoyer l'ensemble $S := \{u : x^*_u \geqslant 1\} \cup \{u \notin I : x^*_u \geqslant 1/2\}$.

[*Exercice*. Montrez que l'algorithme précédent est une 3/2-approximation.]

Pour Couverture par sommets dans le cas planaire il existe même des $(1+\varepsilon)$-approximations pour toute constante $\varepsilon > 0$. Mais attention ! Pour que le temps de ces algorithmes soit polynomial, ε ne peut pas être aussi petit que l'on veut. [*Question*. Quelle valeur d'$\varepsilon > 0$ faudrait-il prendre pour obtenir un algorithme exact ?] En effet, [HC06] ont montré qu'il faut $\varepsilon = \Omega(1/\log^2 n)$. Plus précisément, sauf si P=NP, aucun algorithme avec un facteur d'approximation $1+\varepsilon$ a une complexité $2^{o(1/\sqrt{\varepsilon})} n^{O(1)}$.

De manière générale, un algorithme d'approximation qui, pour chaque constante $\varepsilon > 0$, produit un facteur d'approximation $1 \pm \varepsilon$ en temps polynomial est appelé *schéma d'approximation en temps polynomial* (ou PTAS pour *Polynomial-Time Approximation Scheme* en Anglais). On parle de schéma car l'algorithme pourrait être différent pour chaque ε.

[*Exercice*. Comment modifier (S1) pour la version valuée de Couverture par sommets, c'est-à-dire trouver une couverture de poids minimum, si chaque sommet u possède un poids $\omega_u \geqslant 0$?]

[*Exercice*. Comment modifier l'algorithme PlanarApprox_VC pour la version valuée ?]

3.4 Couverture par ensembles

Le problème Couverture par ensembles (*Set Cover* en Anglais) généralise Couverture par sommets. L'intérêt est qu'il capture beaucoup d'autres problèmes, dont Ensemble dominant. Avec son algorithme glouton associé, il forme un élément essentiel en Algorithmique, comme l'est la formule $E = mc^2$ en Physique.

Couverture par ensembles

Instance: deux ensembles B et $\mathcal{F} \subseteq 2^B$ 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 $\bigcup_{S \in \mathcal{C}} S = B$.

On utilise ici la notation « 2^X » pour représenter l'ensemble des parties d'un ensemble X (*power set* en Anglais). L'ensemble $\mathcal{F}$ est donc une famille de sous-ensembles de B, l'ensemble des éléments à couvrir. 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 $\mathscr{C}$, en particulier $\mathscr{C} = \mathscr{F}$, et donc une plus petite que les autres.

Il existe une version plus générale du problème où chaque ensemble $S \in \mathscr{F}$ possède un coût, qu'on notera coût(S). Il s'agit alors de trouver une couverture $\mathscr{C}$ de B de coût minimum, le coût de $\mathscr{C}$ étant défini par coût($\mathscr{C}$) := $\sum_{S\in\mathscr{C}}$ coût(S). Pour que le problème ait un intérêt, on supposera toujours que le coût des ensembles est > 0, puisque sinon on aura toujours intérêt à prendre autant que possibles des ensembles de coûts $\leqslant 0$ qui couvrent des éléments de B sans rien coûter !

Les exemples donnés ci-dessous considèrent que coût(S) = 1 pour tout $S \in \mathscr{F}$ si bien que coût($\mathscr{C}$) = $|\mathscr{C}|$ est simplement le nombre d'éléments de $\mathscr{C}$.

Couverture par sommets d'un graphe G est une instance de ce problème avec $B = E(G)$ et $\mathscr{F} = \{E_u : u \in V(G)\}$ où E_u est l'ensemble des arêtes incidentes au sommet u (voir figure 3.12).

Figure 3.12 – Couverture par sommets est un cas particulier de Couverture par ensembles. Dans cet exemple (à gauche), $B = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ et $\mathscr{F} = \{\{e_1, e_5, e_6\}, \{e_1, e_2, e_5\}, \{e_1, e_3, e_6\}, \{e_2, e_3, e_4\}, \{e_4\}\}$. Et une couverture possible de taille 3 est $\mathscr{C} = \{\{e_1, e_2, e_5\}, \{e_1, e_3, e_6\}, \{e_4\}\}$. Ensemble dominant est aussi un cas particulier (couverture des sommets par des boules de rayon un). Dans cet exemple (à droite), deux boules (en bleues) suffisent.

Ensemble dominant d'un graphe G est aussi une instance de ce problème avec $B = V(G)$ et $\mathscr{F} = \{B(u, 1) : u \in V(G)\}$ est l'ensemble des boules de rayon un (voir figure 3.12). Il en va de même de sa généralisation, le problème Ensemble dominant à distance r discuté au paragraphe 2.12 où il s'agit de dominer tous les sommets à distance au plus r (donc avec $\mathscr{F} = \{B(u, r) : u \in V(G)\}$). Notons que la version valuée est pertinente dans le cas de la modélisation du positionnement d'antennes radio pour couvrir un territoire (voir figure 2.3). En effet, positionner une antenne au sommet d'une montagne va certes couvrir plus de points mais coûter également beaucoup plus cher qu'en vallée. L'opérateur cherche donc généralement une couverture de *coût* minimum.

Couverture par droites est aussi une instance particulière (voir le paragraphe 2.8.2), ainsi que le problème couverture par faces planaire rencontré page 79.

Il existe une variante de Couverture par ensembles appelée Couverture exacte où la couverture doit couvrir une et une seule fois chaque élément de B. Ce problème de décision NP-complet permet de montrer que le problème classique du Voyageur de commerce dans le cas du plan Euclidien est aussi NP-complet, cf. [Pap77]. On peut bien évidemment résoudre le problème du couplage parfait dans un graphe par Couverture

EXACTE. [*Question*. Comment ?] Cependant ce dernier problème est polynomial d'après la discussion du paragraphe 3.2.1.

Le problème de COUVERTURE PAR ENSEMBLES est équivalent à un autre problème appelé HITTING SET : on veut trouver un ensemble $C \subseteq B$, le *hitting set*, de taille minimum, qui intersectent tous les ensembles de $\mathcal{F}$. Cela revient à échanger le rôle de B et $\mathcal{F}$. Notons qu'une instance $(B, \mathcal{F})$ de COUVERTURE PAR ENSEMBLES définit un hypergraphe, objet déjà évoqué page 110, dont l'ensemble des sommets est B et l'ensemble des hyper-arêtes est $\mathcal{F}$. Un *Hitting Set* est aussi appelé *transversal* de l'hyper-graphe $(B, \mathcal{F})$. Si chaque hyper-arête est de taille deux, soit $|e| = 2$ pour tout $e \in \mathcal{F}$, alors l'hyper-graphe $(B, \mathcal{F})$ est simplement un graphe, et un transerval est simplement une couverture par sommets.

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, ce qui revient à calculer un ENSEMBLE INDÉPENDANT. [*Question*. Dans quel graphe ?] Le célèbre théorème de dualité affirme que les solutions réelles optimales pour un problème LP et son dual sont les mêmes, voir [Chv83, chp. 5].

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 en temps polynomial pour ce problème. On ne démontrera pas dans ce cours que tel est bien le cas. On présente l'algorithme dans sa version valuée, où chaque ensemble $S \in \mathcal{F}$ possède un coût(S).

Dans l'algorithme ci-dessous $\mathcal{C}$ est la couverture construite alors que R représente l'ensemble des éléments restant à couvrir. Dans sa version simple où coût$(S) = 1$ pour tout $S \in \mathcal{F}$, l'algorithme consiste simplement à choisir itérativement l'ensemble S qui couvre le plus d'éléments encore non couverts.

De manière générale, lorsqu'il y a des coûts, on cherche à maximiser le ratio nombre d'éléments nouvellement couverts sur le coût de l'ensemble. Par exemple, si l'on a le choix entre : (1) couvrir 5 éléments au coût de 10, ou bien (2) couvrir 3 éléments au coût de 5. On préférera la 2e solution, de ratio 0.6 contre 0.5 pour la 1ère, avec laquelle on s'attend à pouvoir couvrir 5 autres éléments avec un coût total inférieur à la 1ère solution.

Algorithme GreedySetCover($B, \mathscr{F}$)

1. $R := B$ et $\mathscr{C} := \varnothing$
2. Tant que $R \neq \varnothing$ faire
   (a) Sélectionner un ensemble $S \in \mathscr{F}$ tel que $|S \cap R|/\text{coût}(S)$ est maximum [8]
   (b) $\mathscr{C} := \mathscr{C} \cup \{S\}$ et $R := R \setminus S$
3. Renvoyer $\mathscr{C}$

Dans la pratique on peut, à chaque itération, supprimer S de la famille $\mathscr{F}$ pour éventuellement accélérer la recherche du prochain S. Pour analyser les performances de cet algorithme, on aura besoin de la fonction $H(n)$ donnant le n-ième nombre de la série harmonique définie par :

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

Graphiquement, il est facile de voir que, pour tout entier $n \geqslant 1$,

$$\int_1^{n+1} \frac{1}{x}dx \;<\; H(n) \;\leqslant\; 1 + \int_2^{n+1} \frac{1}{x-1}dx = 1 + \int_1^{n} \frac{1}{x}dx$$

ce qui, en se souvenant que la dérivée $(\ln x)' = 1/x$, implique

$$[\ln x]_1^{n+1} = \ln(n+1) \;<\; H(n) \;\leqslant\; 1 + \ln n \;= 1 + [\ln x]_1^n \tag{3.2}$$

$\ln(1+n) < H(n) \leqslant$ [9] $1 + \ln n$. On a aussi l'assymptotique

$$H(n) \;=\; \gamma + \ln n + \frac{1}{2n} - O\left(\frac{1}{n^2}\right)$$

où $\gamma = 0.5772156649\ldots$ est la constante d'Euler-Mascheroni.

Théorème 3.3 *L'algorithme GreedySetCover($B, \mathscr{F}$) est une $H(k)$-approximation pour* Couverture par ensembles *de coût minimum, où $k = \max_{S \in \mathscr{C}^*} |S|$ et $\mathscr{C}^*$ une solution optimale.*

Notez que $\mathscr{C}^*$, dans la définition de k, peut être choisie arbitrairement parmi les solutions optimales, comme par exemple celle qui minimise k. Cependant, en pratique on ne connaît évidemment pas la solution optimale. Sachant que $\mathscr{C}^* \subseteq \mathscr{F}$, on peut quand même donner un majorant sur k plus simple à calculer. Il vient, en posant $n = |B|$:

$$k \;\leqslant\; \max_{S \in \mathscr{F}} |S| \;\leqslant\; |B| \quad \text{et donc} \quad H(k) \;\leqslant\; 1 + \ln|B| \;=\; 1 + \ln n\,.$$

8. On présente parfois la sélection de S en 2(a) par « coût$(S)/|S \cap R|$ est minimum » mais c'est moins intuitif lorsque les coûts sont unitaires.

9. Il ne peut avoir égalité que si $n = 1$.

Preuve. À chaque boucle de la ligne 2, l'ensemble R représente l'ensemble des éléments restant à couvrir. Comme $\bigcup_{S\in\mathscr{F}} S = B$, il existe donc toujours un ensemble $S \in \mathscr{F}$ 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 trouvent dans $\mathscr{C}$ couvrent tous les éléments de B. Donc $\mathscr{C}$ est bien une couverture par ensembles.

Il est évident que l'algorithme a une complexité en temps qui est polynomiale en la taille de l'instance, c'est-à-dire en $|B| + |\mathscr{F}|$. Notez que la taille de $\mathscr{F}$ n'est pas nécessairement proportionnelle ou polynomiale à celle de B. Il est même possible que $|\mathscr{F}| = 2^{\Omega(|B|)}$, si on a presque tous les sous-ensembles de B dans $\mathscr{F}$ par exemple. Il n'empêche que l'algorithme reste de complexité polynomiale. Il nous reste à montrer que le facteur d'approximation est $H(k)$ avec $k = \max_{S\in\mathscr{C}^*} |S|$.

On représente l'instance de Couverture par ensembles par un graphe biparti. Sur un premier niveau on place les ensembles de $\mathscr{F}$, et sur le second les éléments de B. On met une arête entre $S \in \mathscr{F}$ et $b \in B$ si $b \in S$ comme illustré sur la figure 3.13(a).

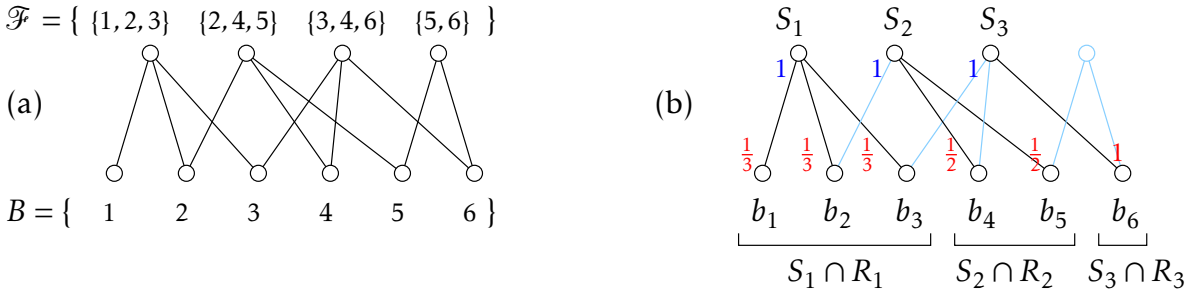


Figure 3.13 – (a) Représentation d'une instance $(B, \mathscr{F})$ de Couverture par ensembles. (b) Coûts des ensembles (en bleu) et prix des éléments (en rouge) lors d'une exécution de l'algorithme glouton.

On suppose une exécution de l'algorithme glouton (cf. figure 3.13(b)), et l'on note S_i l'ensemble sélectionné par l'algorithme à l'étape $i \in \{1,\dots,|\mathscr{C}|\}$, $\mathscr{C}$ 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 coût($\mathscr{C}$), le coût de la couverture renvoyée par l'algorithme. La première est de sommer le coût des ensembles S_i du premier niveau. La seconde est de répartir équitablement, à chaque ensemble S_i sélectionné, 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 couvert par S_i. Formellement, le prix d'un élément lors d'une exécution de l'algorithme est défini par :

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

En quelque sorte, lorsque l'algorithme sélectionne S_i, il « achète » les $|S_i \cap R_i|$ nou-

veaux éléments au prix unitaire moyen de coût$(S_i)/|S_i \cap R_i|$, ce qui va lui coûter un total précisément de coût(S_i).

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

$$\text{coût}(\mathscr{C}) = \sum_{S \in \mathscr{C}} \text{coût}(S) = \sum_{b \in B} \text{prix}(b) \,.$$

Soit $\mathscr{C}^*$ une couverture de coût minimum. Comme chaque élément $b \in B$ est couvert par un ensemble $S \in \mathscr{C}^*$, on a aussi :

$$\sum_{b \in B} \text{prix}(b) \leqslant \sum_{S \in \mathscr{C}^*} \sum_{b \in S} \text{prix}(b) \,.$$

Notez bien que prix(b) est fixé par l'exécution de l'algorithme. Dans l'exemple 3.13, si $\mathscr{C}^* = \{S_1, S_3, \{5,6\}\}$, alors l'inéquation précédente énonce simplement que

$$3 = \underbrace{\left(\frac{1}{3}+\frac{1}{3}+\frac{1}{3}\right)}_{b_1\ b_2\ b_3} + \underbrace{\left(\frac{1}{2}+\frac{1}{2}\right)}_{b_4\ b_5} + \underbrace{\left(1\right)}_{b_6} \leqslant \underbrace{\left(\frac{1}{3}+\frac{1}{3}+\frac{1}{3}\right)}_{b_1\ b_2\ b_3} + \underbrace{\left(\frac{1}{3}+\frac{1}{2}+1\right)}_{b_3\ b_4\ b_6} + \underbrace{\left(\frac{1}{2}+1\right)}_{b_5\ b_6} = 4+\frac{1}{3} \,.$$

Considérons un élément b_j qui devient couvert à l'étape i par l'ensemble S_i. Notez que $b_j \in R_i$. On souhaite majorer prix(b_j) en fonction du coût d'un ensemble $S \in \mathscr{C}^*$.

On remarque que l'algorithme, en choisissant l'ensemble S_i qui maximise $|S_i \cap R_i|/\text{coût}(S_i)$, minimise le prix des éléments à couvrir. Dit autrement,

$$\text{prix}(b_j) = \frac{\text{coût}(S_i)}{|S_i \cap R_i|} \leqslant \frac{\text{coût}(S)}{|S \cap R_i|} \qquad \forall S \ni b_j \,.$$

Afin de majorer prix(b_j), on minore $|S \cap R_i|$ par une expression indépendante de R_i. Comme $b_j \in R_i$, tous les éléments sélectionnés plus tard sont aussi dans R_i, et donc $b_j, b_{j+1}, \ldots, b_n \in R_i$. Et comme $b_j \in S$, les éléments de S d'indices $\geqslant j$ sont également dans R_i. Par exemple :

$$\begin{aligned} b_j \in R_i &= \{\ldots, b_j, \ldots, b_{j+3}, \ldots, b_{j+7}, \ldots, b_n\} \\ b_j \in S &= \{\ldots, \underbrace{b_j, b_{j+3}, b_{j+7}}_{\sup_S(b_j)}\} \end{aligned}$$

Il suit que $|S \cap R_i|$ est au moins égale au nombre d'éléments de S d'indices $\geqslant j$. Notons $\sup_S(b_j)$ ce nombre. Plus formellement, pour tout $b \in S$, notons $\sup_S(b)$ le nombre d'éléments de S d'indice supérieur ou égale à celui de b. D'où $|S \cap R_i| \geqslant \sup_S(b_j)$.

Maintenant, pour tout $S \in \mathscr{F}$ et $b \in S$, il existe des indices i, j tels que $b = b_j \in R_i$, et donc

$$\text{prix}(b) \leqslant \frac{\text{coût}(S)}{\sup_S(b)} \,.$$

En combinant les équations précédentes, on obtient :

$$\text{coût}(\mathscr{C}) \leqslant \sum_{S\in\mathscr{C}^*}\sum_{b\in S}\text{prix}(b) \leqslant \sum_{S\in\mathscr{C}^*}\sum_{b\in S}\frac{\text{coût}(S)}{\sup_S(b)} \leqslant \sum_{S\in\mathscr{C}^*}\text{coût}(S)\cdot\left(\sum_{b\in S}\frac{1}{\sup_S(b)}\right).$$

On remarque que si les indices des éléments b de S sont $j_1 < j_2 < \cdots < j_{|S|}$, soit $S = \{b_{j_1},\ldots,b_{j_{|S|}}\}$, alors $\sup_S(b_{j_1}) = |S|$, $\sup_S(b_{j_2}) = |S|-1$, ... et $\sup_S(b_{j_{|S|}}) = 1$. Et donc,

$$\sum_{b\in S}\frac{1}{\sup_S(b)} = \frac{1}{|S|}+\frac{1}{|S|-1}+\cdots+\frac{1}{1} = H(|S|)\,.$$

Il suit, en posant $k = \max_{S\in\mathscr{C}^*}|S|$, que :

$$\text{coût}(\mathscr{C}) \leqslant \sum_{S\in\mathscr{C}^*}\text{coût}(S)\cdot H(|S|) \leqslant \left(\sum_{S\in\mathscr{C}^*}\text{coût}(S)\right)\cdot H(k) \leqslant \text{coût}(\mathscr{C}^*)\cdot H(k)\,.$$

L'algorithme glouton est donc une $H(k)$-approximation. □

Coût unitaire. Si les coûts vérifient $\text{coût}(S)\in]0,1]$, en particulier s'ils sont unitaires, on peut montrer (cf. [You08]) que l'algorithme glouton renvoie une couverture de coût au plus :

$$\lceil\text{coût}(\mathscr{C}^*)\rceil + \text{coût}(\mathscr{C}^*)\cdot(H(n) - H(\lceil\text{coût}(\mathscr{C}^*)\rceil))$$

soit un facteur d'approximation inférieur à [10]

$$1+\ln\left(\frac{n}{\text{coût}(\mathscr{C}^*)}\right)+\frac{1}{\text{coût}(\mathscr{C}^*)}\,.$$

Cela qui donne une approximation constante lorsque le coût de la solution optimale est linéaire en $n = |B|$. Par exemple, dans un graphe de diamètre $D > 0.03n$, on peut approximer la taille du plus petit ensemble dominant (nécessairement de taille $> D/3 = n/100$ d'après le lemme 2.5) avec un facteur au plus $1+\ln(100)+100/n < 5.7$ pour n suffisamment grand. C'est mieux que la solution consistant à prendre un ensemble dominant de taille $\lfloor n/2\rfloor$, qui existe toujours [*Question*. Pourquoi ?], et qui donnerait un facteur d'approximation d'au plus $(n/2)/(n/100) = 50$.

Facteur d'approximation. La valeur $H(n)$ n'est pas le meilleur facteur d'approximation pour l'algorithme glouton. Dit autrement, le théorème 3.3 ne donne pas la meilleure analyse possible de cet algorithme en fonction de $n = |B|$. Il a été démontré

10. D'après les inégalités (3.2), $H(x) - H(y) < 1 + \ln(x/y)$.

dans [Sla96] que le facteur d'approximation était $\ln n - \ln\ln n + O(1)$. Plus précisément, le facteur d'approximation $\alpha(n)$ de l'algorithme glouton vérifie

$$-0.31 \;<\; -1+\ln 2 \;<\; \alpha(n) - (\ln n - \ln\ln n) \;<\; 3 - \ln 32 + \ln\ln 32 \;<\; 0.78\,.$$

Il a été montré [Fei98][DS13] qu'aucun algorithme polynomial, sauf si P=NP, ne peut donner un facteur d'approximation inférieur à $(1-o(1))\ln n$. Cela montre au passage qu'il doit exister des instances où l'algorithme glouton produit un facteur d'approximation d'au moins $(1-o(1))\ln n$, enfin sauf si P=NP.

Instance critique. Essayons de construire des instances critiques pour l'algorithme glouton. Cet exemple apparaît déjà dans [BG95], qui serait d'après les auteurs du à Jiří Matoušek. On considère l'ensemble d'éléments $B = S_1 \cup \cdots \cup S_t = T_0 \cup T_1$ et la famille $\mathcal{F} = \{S_1, \ldots, S_t, T_0, T_1\}$ où les unions sont disjointes. De plus chaque sous-ensemble S_i a 2^i éléments, T_0 contient la moitié de chacun des éléments des S_i, et T_1 l'autre moitié. L'ensemble B comprend $n = |B| = \sum_{i=1}^{t} 2^i = 2^{t+1} - 2$ éléments. Notons que $t = \log_2(n+2) - 1 > (\log_2 n) - 1$. Voir la figure 3.14 pour $t = 4$.

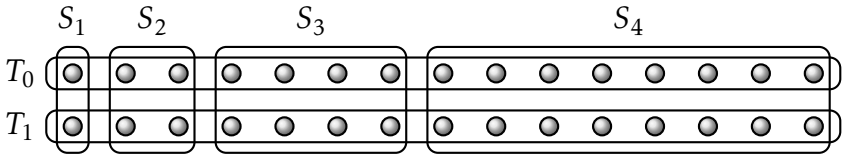


FIGURE 3.14 – Instance presque critique pour l'algorithme glouton.

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. Donc S_{t-1} sera choisi. Et ainsi de suite. Au final, l'algorithme renvoie une couverture de taille $t > (\log_2 n) - 1$, 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.

Pour rendre « plus critique » une instance, il faudrait considérer des coûts non uniformes, et surtout être sûr que $H(n)$ est bien le facteur d'approximation de l'algorithme glouton.

[*Exercice*. On propose la règle de réduction suivante, avant l'application de l'algorithme glouton : « S'il existe un ou plusieurs éléments de B couverts par un unique ensemble $S \in \mathcal{F}$, mettre S dans la couverture, et recommencer avec $(B \setminus S, \mathcal{F})$. Sinon, appliquer GreedySetCover$(B, \mathcal{F})$. » L'intuition est que la variante ne peut être que meilleure puisque la couverture optimale doit contenir S. (1) Montrez cependant qu'en ajoutant 5 éléments et un ensemble à l'instance critique vue précédemment, la variante devient $\Omega(\log n)$ fois plus mauvaise que l'algorithme glouton ! (2) Montrez que l'autre variante de cette réduction, consistant à prendre S mais à ne supprimer dans B que les éléments

uniquement couverts, se trouve encore être $\Omega(\log n)$ fois plus mauvaise que l'algorithme glouton.]

Ensemble dominant. Pour Ensemble dominant, l'algorithme glouton peut, pour tout entier $t \geqslant 1$, renvoyer un ensemble dominant de taille $4t$, alors qu'un ensemble de taille $3t$ 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 2.14). L'algorithme glouton renverra pour chaque composante 4 au lieu de 3, soit une approximation de $4/3 \approx 1.33$. Pour les graphes de degré $\leqslant 3$, comme ce contre-exemple, l'algorithme glouton fournit de manière générale une $H(4)$-approximation puisque les ensembles sont de taille $k \leqslant 4$. Et $H(4) = 1+1/2+1/3+1/4 = 25/12 \approx 2.08$. Ce graphe n'est donc pas une instance critique pour les graphes de degrés $\leqslant 3$. En utilisant le meilleur algorithme pour 4-*Set Cover* (voir le paragraphe ci-après), le facteur aurait été de $H(4) - 0.5025 \approx 1.58$.

Notons que l'étoile subdivisée est aussi un contre-exemple pour Couverture par sommets, avec le même facteur d'approximation.

[*Exercice*. Modifiez l'exmple critique de la figure 3.14 pour en déduire un exemple pour Ensemble dominant où l'algorithme glouton produit un facteur d'approximation $\Omega(\log n)$.]

Lorsque *k* est petit. D'autres algorithmes un peu meilleurs que l'algorithme glouton existent pour résoudre Couverture par ensembles lorsque k, ici la taille maximum des ensembles de $\mathcal{F}$, est « petit ». On parle du problème k-Couverture par ensembles. Les meilleures algorithmes donnent les facteurs d'approximation suivants, tous de la forme $H(k) - c_k$ pour un certain réel $c_k \geqslant 1/2$ (voir [ACK09]) :

- $H(k) - 0.5$ pour tout $k \geqslant 2$;
- $H(k) - 0.5025...$ pour tout $k \geqslant 4$;
- $H(k) - 0.5291...$ pour tout $k \geqslant 6$;
- $H(k) - 0.5904...$ pour k assez grand.

Ainsi, pour $k = 2$, ces algorithmes fournissent un algorithme exact polynomial car $H(2) - 0.5 = 1 + 1/2 - 0.5 = 1$. [*Exercice*. Donnez un problème correspondant à $k = 2$ ou 2-Couverture par ensembles ?] Et pour $k = 3$, le facteur d'approximation est $H(3) = 1 + 1/2 + 1/3 - 0.5 = 4/3$. L'idée de ces algorithmes est de gérer différemment le cas où les ensembles ne couvrent que peu d'éléments encore couverts, disons 2, 3 ou 4 éléments. Par exemple, lorsque l'algorithme ne couvre plus que deux éléments à la fois, on peut essayer de trouver un couplage maximum de coût minimum (ce qui est polynomial) dans le graphe où l'on connecterait des éléments non couverts qui peuvent l'être par un ensemble de $\mathcal{F}$. Cela ne peut qu'être meilleur que de prendre un couplage maximal, ce que fait l'algorithme glouton.

Il a été montré que pour k-Couverture par ensembles, le facteur d'approximation doit être au moins $\ln k - \Omega(\ln \ln k)$, sauf si P=NP [Tri01]. L'algorithme glouton est donc,

d'une certaine façon, quasiment le meilleur possible.

Lorsque la VC-dimension est bornée. D'autres algorithmes, comme celui de [BG95], ont été proposé lorsque la VC-dimension du système $(B,\mathcal{F})$ est petite (voir page 110). Plus précisément, si la VC-dimension du dual[11] $(\mathcal{F},B^*)$ est d, alors on peut résoudre Couverture par sommets pour $(B,\mathcal{F})$ en temps polynomial avec un facteur d'approximation $O(d\log(d\cdot \text{OPT}))$. En particulier, cela fait $O(\log \text{OPT})$ si d est bornée. Il est connue que la VC-dimension de $(B,\mathcal{F})$ est au plus 2^{d+1}. Notons que l'exemple précédant, critique pour GreedySetCover, a pour VC-dimension de son dual $d \leqslant 2$, car chaque élément n'est couvert qu'au plus deux fois. L'algorithme de [BG95] donne un facteur constant (en fait, il donne l'optimal) au lieu de $\Omega(\log n)$ comme pour GreedySetCover.

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 3.13), cela correspondant au degré maximum d'un sommet de B.

Plus formellement, la fréquence est définie par :

$$f(B,\mathcal{F}) \;=\; \max_{b\in B} |\mathcal{S}(b)| \qquad \text{où} \quad \mathcal{S}(b) = \{S\in\mathcal{F} : b\in S\}$$

est la famille des ensembles de $\mathcal{F}$ contenant b. Toute instance de Couverture par sommets a une fréquence $f = 2$. Chaque arête $uv \in B$ apparaît dans exactement deux ensembles, E_u et E_v. Pour Ensemble dominant la fréquence est $f \leqslant \Delta + 1$, un de plus que le degré maximum du graphe.

L'algorithme suivant est une variante de l'algorithme glouton précédent dans le cas des coûts unitaires.

Algorithme SetCoverFreq$(B,\mathcal{F})$

1. $R := B$ et $\mathcal{C} := \varnothing$
2. Tant qu'il existe $b \in R$ faire $\mathcal{C} := \mathcal{C} \cup \mathcal{S}(b)$ et $R := R \setminus \bigcup_{S\in\mathcal{S}(b)} S$
3. Renvoyer $\mathcal{C}$

Théorème 3.4 *L'algorithme SetCoverFreq$(B,\mathcal{F})$ est une f-approximation pour Couverture par ensembles de taille minimum avec coûts unitaires, où $f = f(B,\mathcal{F})$ est la fréquence de l'instance.*

11. Dans la représentation en graphe biparti, cela revient à inverser les deux parties : les éléments à couvrir sont les ensembles de $\mathcal{F}$, et pour le faire il faut utiliser des éléments de $B^* = \{\mathcal{S}(b) : b\in B\}$ où $\mathcal{S}(b)$ est la famille des ensembles de $\mathcal{F}$ contenant b. On y reviendra au paragraphe 3.4.2.

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

Soient b_i l'élément de B choisi à la i-ème itération de l'étape 2, et t le nombre d'itérations. Comme $|\mathcal{S}(b_i)| \leqslant f$, il suit que

$$|\mathcal{C}| \;=\; \sum_{i=1}^{t} |\mathcal{S}(b_i)| \;\leqslant\; tf\ .$$

Notons que pour chaque i, la famille $\mathcal{S}(b_i)$ contient au moins un ensemble $S_i^* \in \mathcal{C}^*$ de la couverture optimale, sinon l'élément b_i ne pourrait pas être couvert par $\mathcal{C}^*$. De plus $S_i^* \neq S_j^*$, car les familles $\mathcal{S}(b_i)$ et $\mathcal{S}(b_j)$ sont deux-à-deux disjointes. Par conséquent $|\mathcal{C}^*| \geqslant t$, et donc

$$|\mathcal{C}| \;\leqslant\; tf \;\leqslant\; f|\mathcal{C}^*|\ .$$

□

[*Exercice*. Donnez, pour chaque f, une instance critique pour cet algorithme.]

Comme on l'a déjà dit, pour Ensemble dominant, $f \leqslant \Delta + 1$ et donc l'approximation de l'algorithme GreedySetCover est meilleure car $H(\Delta+1) < \Delta + 1$ dès que $\Delta \geqslant 1$. Mais pour Couverture par sommets où $f = 2$, l'algorithme SetCoverFreq donne l'approximation standard et un algorithme très proche (sinon le même) de l'Approx_VC1 vu au paragraphe 3.3.1. [*Exercice*. Quel aurait été la le facteur d'approximation pour l'exemple de la figure 3.14 si on avait utilisé SetCoverFreq ?]

En fait, l'algorithme SetCoverFreq continue de fonctionner si, au lieu d'utiliser la famille $\mathcal{S}(b)$, on utilise $\overline{\mathcal{S}}(b)$ définie comme *une* famille d'ensembles de $\mathcal{F}$ dont l'union contient celle de $\mathcal{S}(b)$, union notée ci-après $U(b)$. C'est-à-dire qu'il s'agit de prendre toute famille $\overline{\mathcal{S}}(b)$ telle que

$$\bigcup_{S \in \overline{\mathcal{S}}(b)} S \;\supseteq\; \bigcup_{S \in \mathcal{S}(b)} S = U(b).$$

En effet, le nombre d'itérations de cette variante ne peut pas être pire que celui de SetCoverFreq. [*Exercice*. Montrez que cela est bien vrai malgré le fait que dans la preuve du théorème 3.4 les mêmes arguments ne s'appliquent pas à $\overline{\mathcal{S}}(b)$ comme à $\mathcal{S}(b)$. Il est faux par exemple de dire que $\overline{\mathcal{S}}(b_i)$ contient toujours un ensemble de la couverture optimale couvrant b_i.] De plus, si pour chaque b, $\left|\overline{\mathcal{S}}(b)\right| \leqslant |\mathcal{S}(b)|$, la taille de la couverture renvoyée ne pourra être que plus petite. Et, le facteur d'approximation ne pourra être que meilleur. Plus précisément, il sera au plus de

$$\overline{f} = \max_{b \in B} \left|\overline{\mathcal{S}}(b)\right| \;\leqslant\; \max_{b \in B} |\mathcal{S}(b)| = f\ .$$

On pourrait donc choisir pour $\overline{\mathcal{S}}(b)$ la plus petite famille couvrant $U(b)$. Malheureusement, c'est résoudre le problème Couverture par ensembles sur l'instance $(U(b), \mathcal{S}(b))$

pour chaque b. On tourne en rond ... Cependant pour Ensemble dominant on pourrait choisir $\overline{\mathscr{S}}(u) = \bigcup_{v \in N(u)} B(v,1) = \mathscr{S}(u) \setminus \{B(u,1)\}$, donc avec $\left|\overline{\mathscr{S}}(u)\right| = \deg(u) \leqslant \Delta$. Le facteur d'approximation devient alors Δ au lieu de $\Delta+1$. Malheureusement, cela n'est pas mieux que l'algorithme GreedySetCover car $H(\Delta+1) < \Delta$ lorsque [12] $\Delta \geqslant 2$.

Il existe une borne inférieure de $f^{1/19}$ sur le facteur d'approximation de tout algorithme d'approximation de Couverture par sommets sur des instances de fréquence f, sauf si P=NP (cf. [Tri01]).

3.4.3 Plus encore

[*Cyril*. Section à finir.]

LP pour Couverture par ensembles. Dans cette formulation, chaque ensemble $S \in \mathscr{F}$ reçoit une variable $x_S \in \{0,1\}$ telle que $x_S = 1$ si S est choisi pour être dans la couverture, et $x_S = 0$ sinon. On souhaite minimiser le coût de la couverture, soit le nombre de variables à 1. Cela revient à minimiser la somme de toutes les variables (pondérées par leurs coûts), avec bien sûr la contrainte que tout élément $b \in B$ doit être couvert, c'est-à-dire que la somme des x_S pour $b \in S$ doit être au moins 1. Cela donne,

$$\begin{aligned} & \min \sum_{S \in \mathscr{F}} x_S \cdot \text{coût}(S) && && \text{(S3)} \\ \text{tel que :} \quad & \textstyle\sum_{S \in S(b)} x_S \geqslant 1 && \forall b \in B \\ & x_S \in \{0,1\} && \forall S \in \mathscr{F} \end{aligned}$$

Dans le cas des coûts uniformes, on peut faire la relaxation classique consistant à remplacer la contrainte discrete $x_S \in \{0,1\}$ par $x_S \in [0,1]$, soit $x_S \geqslant 0$. À partir d'une solution optimale x^* de la relaxation du système (S3), on peut renvoyer $\mathscr{C} = \left\{x_S^* : x_S^* \geqslant 1/f\right\}$, où $f = f(B,\mathscr{F})$ est la fréquence de l'instance. On peut alors montrer qu'il s'agit d'une f-approximation tout comme l'algorithme SetCoverFreq.

***k*-Couverture Maximum.** Dans cette variante de Couverture par ensembles (on parle de *Maximum k-Coverage* en Anglais), il s'agit avec k ensembles pris dans $\mathscr{F}$ de couvrir un nombre maximum d'éléments de B. Cette variante assez naturelle considère des coûts unitaires. On peut alors montrer qu'en répétant k étapes de l'algorithme GreedySetCover le facteur d'approximation est de $1-(1-1/k)^k \geqslant 1-e^{-1} \approx 0.63$, voir [Fei98][Proposition 5.1].

12. Pour $\Delta = 2$, $H(\Delta+1) = H(3) = 11/6 < 2$.

Oracle. Pour Couverture par ensembles il arrive souvent que $\mathcal{F}$ soit de taille exponentielle en $|B|$, voir même de taille infinie et donné de manière implicite. C'est le cas par exemple lorsqu'on veut couvrir un ensemble $B \subseteq \mathbb{R}^2$ de n points par des disques de rayon unité, Couverture par disques unités (ou *Unit Disk Cover* en Anglais). Ici $\mathcal{F}$ est infini puisque tous les disques de rayon unité du plan pourraient être considérés. Pour pouvoir continuer à utiliser l'algorithme GreedySetCover il est raisonable de supposer qu'on dispose d'une procédure (un oracle) qui, à chaque étape i, étant donné un ensemble R_i d'éléments restant à couvrir, puisse fournir l'ensemble $S_i \in \mathcal{F}$ de meilleur coût, peut-être à un facteur α près.

Pour Couverture par disques unités, qui est NP-hard [13], il est possible de calculer le meilleur disque en temps polynomial, donc d'avoir un oracle avec $\alpha = 1$. En effet, il est facile de vérifier [14] qu'à tout disque unité optimal D^* on peut associer (par glissement et contraction de D^*) un autre disque D de rayon au plus 1, couvrant les mêmes points que D^* mais avec deux ou trois points sur son bord. Et si c'est deux, les points sont sur le diamètre de D. Il suit un algorithme *brute-force* en $O(n^4)$ en testant toutes les $\binom{n}{2}$ paires et tous les $\binom{n}{3}$ triplets de points.

Dans le cas d'un oracle avec un facteur d'approximation α quelconque, on peut alors montrer que l'algorithme glouton fournit une $\alpha H(k)$-approximation et aussi une $(1-(1-1/t)^{t/\alpha})$-approximation pour le problème t-Couverture Maximum. [*Cyril.* A-t-on un résultat similaire pour l'approximation de $\mathcal{S}(b)$ pour SetCoverFreq?]

Quant au problème Couverture par disques unités, il existe de meilleurs algorithmes, plus performants que l'algorithme glouton avec un oracle optimal. Par exemple, il existe une 4-approximation de complexité $O(n \log n)$, mais aussi des $2(1+\varepsilon)$-approximation de complexité $O(n^7/\varepsilon^2)$ (voir [BLMS17]) avec des difficultés d'implémentation variables.

3.4.4 Un algorithme exact

On va présenter un algorithme exact pour Couverture par ensembles lorsque les coûts sont unitaires. Notons que ce cas particulier permet de résoudre des problèmes comme Ensemble dominant de taille minimum. L'analyse de sa complexité, qui est exponentielle, ne sera pas abordée. Elle fait appel à des techniques de mesures non triviales pour les algorithmes récursifs.

Cet algorithme va être l'occasion de présenter un algorithme exact dans le sous-cas particulier où tous les ensembles de la famille $\mathcal{F}$ sont de taille deux. L'entrée $(B, \mathcal{F})$ peut alors être vue comme un graphe $G = (B, \mathcal{F})$. Le problème Couverture par ensembles pour G devient alors le problème Couverture par arêtes qui consiste à couvrir tous les sommets ayant au moins un voisin par des arêtes. (Voir la figure 3.15 pour un exemple

13. Reste difficile pour les normes ℓ_1, ℓ_2 et ℓ_∞.
14. Cette observation est due à une communication privée avec Ahmad Biniaz (2020) que je remercie.

d'exécution.) Contrairement à Couverture par sommets, on sait résoudre Couverture par arêtes de taille minimum en temps polynomial.

Algorithme EdgeCoverMin(V, E)

1. Calculer un couplage maximum M.
2. Tant qu' $\exists u \in V$, $\deg(u) \geqslant 1$, non couvert par M, choisir une arête $uv \in E$ et l'ajouter à M.
3. Renvoyer M.

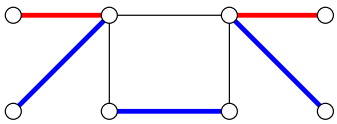


Figure 3.15 – L'algorithme EdgeCoverMin. En bleu les arêtes du couplage maximum, et en rouge celles couvrant l'ensemble indépendant restant.

La validité de cet algorithme est basé sur : (1) couvrir un sous-ensemble $X \subseteq V$ nécessite $|X|/2$ arêtes ; (2) l'ensemble X des sommets couverts par un couplage M vérifie précisément $|X| = 2|M|$; (3) si M est maximal (en particulier s'il est maximum), $V \setminus X$ est un ensemble indépendant ; et (4) couvrir un ensemble indépendant I nécessite $|I|$ arêtes. [*Question*. Pourquoi ces 4 points sont véridiques ?]

L'intuition, mais cela n'est pas une preuve formelle, est qu'il faut minimiser la taille de l'ensemble indépendant $|I| = |V \setminus X| = |V| - 2|M|$, ce qui revient donc à maximiser M.

Algorithme SetCoverMin($B, \mathcal{F}$)

1. Simplifier $\mathcal{F}$ en posant $\mathcal{F} := \{S \cap B : S \in \mathcal{F}\}$.
2. Si $|\mathcal{F}| = \varnothing$, renvoyer 0.
3. Si $\exists S, S' \in \mathcal{F}$ avec $S \subsetneq S'$, renvoyer SetCoverMin($B, \mathcal{F} \setminus \{S\}$).
4. Si $\exists v \in B$ et un unique $S \in \mathcal{F}$ avec $v \in S$, renvoyer $1 +$ SetCoverMin($B \setminus S, \mathcal{F}$).
5. Choisir $S \in \mathcal{F}$ de plus grande taille.
6. Si $|S| = 2$, renvoyer EdgeCoverMin($B, \mathcal{F}$).
7. Renvoyer $\min\{$SetCoverMin($B, \mathcal{F} \setminus \{S\}$), $1 +$ SetCoverMin($B \setminus S, \mathcal{F}$)$\}$.

On remarquera qu'à l'étape 6, et si $|S| = 2$, la famille $\mathcal{F}$ ne comprend que des ensembles de taille deux, ce qui justifie l'emploi de EdgeCoverMin. [*Exercice*. Justifier les 6 premières étapes de l'algorithme.] L'idée est que dans la récurrence principale (étape 7), l'ensemble S est de grande taille (au moins 3), et donc on diminue fortement soit B soit $\mathcal{F}$. La somme $k = |B| + |\mathcal{F}|$ diminue d'1 ou de 3. Cela donne *a priori* une complexité exponentielle (à des polynômes près) en $c^k \approx 1.4655^k = 2^{0.551 \cdot (|B|+|\mathcal{F}|)}$ d'après l'équation 2.1 dans l'analyse de la récurrence du théorème 2.5. Seulement, cette analyse n'est pas fine.

Il a été montré que la complexité de cet algorithme est de $2^{0.305 \cdot (|B|+|\mathcal{F}|)} \cdot (|B| + |\mathcal{F}|)^{O(1)}$. En utilisant une technique de mémorisation, nécessitant un espace $2^{O(|B|+|\mathcal{F}|)}$ (et donc exponentiel), on peut encore réduire sa complexité à $2^{0.299 \cdot (|B|+|\mathcal{F}|)} \cdot (|B| + |\mathcal{F}|)^{O(1)}$.

La complexité exacte de cet algorithme n'est pas connue lorsqu'il est l'appliqué à une instance du problème d'Ensemble dominant de taille minimum sur un graphe à n sommets. Elle est dans ce cas au moins $\Omega(3^{n/4}) = \Omega(2^{0.396n})$ (cf. [FGK09]) et au plus $2^{0.598n} \cdot n^{O(1)}$ puisque $|B| + |\mathcal{F}| = 2n$ dans ce cas.

Bibliographie

[ABMP91] H. Alt, N. Blum, K. Mehlhorn, and M. Paul, *Computing a maximum cardinality matching in a bipartite graph in time* $O(n^{1.5}\sqrt{m/\log n})$, Information Processing Letters, 37 (1991), pp. 237–240. doi : 10.1016/0020-0190(91)90195-N.

[ACK09] S. Athanassopoulos, I. Caragiannis, and C. Kaklamanis, *Analysis of approximation algorithms for k-set cover using factor-revealing linear programs*, Theory of Computing Systems, 45 (2009), pp. 555–576. doi : 10.1007/s00224-008-9112-3.

[AS00] N. Alon and J. H. Spencer, *The Probabilistic Method (2nd edition)*, Wiley-Interscience Publication, 2000.

[BG95] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite VC-dimension*, Discrete & Computational Geometry, 14 (1995), pp. 463–479. doi : 10.1007/BF02570718.

[BLMS17] A. Biniaz, P. Liu, A. Maheshwari, and M. Smid, *Approximation algorithms for the unit disk cover problem in 2D and 3D*, Computational Geometry : Theory and Applications, 60 (2017), pp. 8–18. doi : 10.1016/j.comgeo.2016.04.002.

[BYE82] R. Bar-Yehuda and S. Even, *On approximating a vertex cover for planar graphs*, in 14th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 1982, pp. 303–309. doi : 10.1145/800070.802205.

[Chv83] V. Chvátal, *Linear Programming*, W.H. Freeman, 1983.

[DS13] I. Dinur and D. Steurer, *Analytical approach to parallel repetition*, in 46th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 2013, pp. 624–633. doi : 10.1145/2591796.2591884.

[Edm65] J. Edmonds, *Paths, trees, and flowers*, Canadian Journal of Mathematics, 17 (1965), pp. 449–467. doi : 10.4153/CJM-1965-045-4.

[ERV07] M. Englert, H. Röglin, and B. Vöcking, *Worst case and probabilistic analysis of the 2-opt algorithm for the TSP*, in 18th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2007, pp. 1295–1304.

[ET75] S. Even and R. E. Tarjan, *Network flow and testing graph connectivity*, SIAM Journal on Computing, 4 (1975), pp. 507–518. doi : 10.1137/0204043.

[Fei98] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, Journal of the ACM, 45 (1998), pp. 634–652. doi : 10.1145/285055.285059.

[FF56] L. R. Ford and D. R. Fulkerson, *Maximal flow through a network*, Canadian Journal of Mathematics, 8 (1956), pp. 399–404. doi : 10.4153/CJM-1956-045-5.

[FGK09] F. V. Fomin, F. Grandoni, and D. Kratsch, *A measure & conquer approach for the analysis of exact algorithms*, Journal of the ACM, 56 (2009), pp. Article No. 25, pp. 1–32. doi : 10.1145/1552285.1552286.

[GR98] A. V. Goldberg and S. B. Rao, *Beyond the flow decomposition barrier*, Journal of the ACM, 45 (1998), pp. 783–797. doi : 10.1145/290179.290181.

[HC06] X. Huang and J. Chen, *On QTAS for planar graph problems*, in 4th IFIP International Conference on Theoretical Computer Science, G. Navarro, L. Bertossi, and Y. Kohayakawa, eds., vol. 209, Springer, 2006, pp. 299–313. doi : 10.1007/978-0-387-34735-6_24.

[HK73] J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231. doi : 10.1137/0202019.

[Kar73] A. V. Karzanov, *O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh*, in Matematicheskie Voprosy Upravleniya Proizvodstvom, L. Lyusternik, ed., vol. 5, Moscow State University Press, 1973, pp. 81–94.

[Kar04] G. Karakostas, *A better approximation ratio for the Vertex Cover problem*, in Electronic Colloquium on Computational Complexity (ECCC), TR04-084, October 2004. https://eccc.weizmann.ac.il/report/2004/084/.

[Kar15] M. Karpinski, *Towards better inapproximability bounds for TSP : A challenge of global dependencies*, in Electronic Colloquium on Computational Complexity (ECCC), TR15-097, June 2015. https://eccc.weizmann.ac.il/report/2015/097/.

[KRT94] V. King, S. B. Rao, and R. E. Tarjan, *A faster deterministic maximum flow algorithm*, Journal of Algorithms, 17 (1994), pp. 447–474. doi : 10.1006/jagm.1994.1044.

[Mąd13] A. Mądry, *Navigating central path with electrical flows : From flows to matchings, and back*, in 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 2013, pp. 253–262. doi : 10.1109/FOCS.2013.35.

[MS04] M. Mucha and P. Sandowski, *Finding maximum matchings via Gaussian elimination*, in 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 2004, pp. 248–255. doi : 10.1109/FOCS.2004.40.

[Orl13] J. B. Orlin, *Max flows in $O(nm)$ time, or better*, in 45th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, June 2013, pp. 765–774. doi : 10.1145/2488608.2488705.

[Pap77] C. H. Papadimitriou, *The Euclidean travelling salesman problem is NP-complete*, Theoretical Computer Science, 4 (1977), pp. 237–244. doi : 10.1016/0304-3975(77)90012-3.

[Sla96] P. Slavík, *A tight analysis of the greedy algorithm for set cover*, in 28th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 1996, pp. 435–441. doi : 10.1145/237814.237991.

[SV14] A. Sebö and J. Vygen, *Shorter tours by nicer ears : 7/5-Approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs*, Combinatorica, 34 (2014), pp. 597–629. doi : 10.1007/s00493-014-2960-3.

[Tri01] L. Trivisan, *Non-approximability results for optimization problems on bounded degree instances*, in 33rd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, July 2001, pp. 453–461. doi : 10.1145/380752.380839.

[You08] N. E. Young, *Greedy Set-Cover Algorithms*, Springer, 2008, pp. 379–381. doi : 10.1007/978-0-387-30162-4_175.

[Zwi95] U. Zwick, *The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate*, Theoretical Computer Science, 148 (1995), pp. 165–170. doi : 10.1016/0304-3975(95)00022-O.