Introduction à la complexité paramétrée

Frédéric Havet* Christophe Paul†

JCALM Juin 2008

Ces notes ont été rédigées comme support aux journées CALM à Sophia-Antipolis les 5 et 6 juin 2008.

1 Introduction

La théorie de la NP-complétude considère qu'un problème est *soluble* s'il admet un algorithme polynomial pour le résoudre. Mais beaucoup de problèmes sont NP-difficiles et (à moins que $P = NP$), ils n'admettent pas de tels algorithmes. Cependant une fois qu'un problème est montré NP-difficile, que faire si on doit malgré tout le résoudre? Plusieurs méthodologies peuvent être envisagées: algorithmes d'approximation, algorithmes randomisés, heuristiques, algorithmes exponentiels exacts. Chacune de ces méthodes a des avantages et désavantages.

L'algorithmique à paramètre fixée se place dans l'approche des algorithmes exponentiels exacts en proposant une analyse de la complexité bidimensionnelle et donc plus fine. L'idée fondamentale est d'essayer d'obtenir une meilleure vision de la complexité d'un problème en explorant comment certains paramètres spécifiques du problèmes influencent cette complexité. Idéalement, on cherche à trouver un paramètre k tel que si k est petit alors le problème peut être résolu efficacement. Avant de commencer le développement formel de la théorie de la complexité paramétrée (Partie 3), nous examinons dans la Partie 2 quelques exemples qui illustrent bien les différents types d'influence que peuvent avoir des paramètres sur la complexité d'un problème.

2 Influence des paramètres – quelques exemples

2.1 Un problème de décision: Satisfaisabilité

En théorie classique de la complexité, un problème de décision Q est décrit comme un langage sur des alphabets finis Σ. Si Q est un problème de décision sur l'alphabet Σ, on appelle *instance* de Q un mot x de Σ^*. Habituellement, on représente un problème de décision Q de la façon suivante:

Instance: $x \in \Sigma^*$.

Question: Est-ce que $x \in Q$?

Le problème de décision le plus fondamental est le problème SAT de satifaisabilité des formules booléennes en forme normale conjonctive. Une *formule booléenne en forme normale conjonctive* est une conjonction de *clauses*, chaque clause étant une disjonction de *litéraux*, c'est-à-dire des variables booléennes niées ou pas.

Problème 1 (SAT)

Instance: Une formule boolénne F en forme normale conjonctive.

Question: F est-elle satisfaisable? i.e. existe-t-il une affectation de valeur *vrai* ou $faux$ aux variables de manière à ce que F soit vraie?

*Projet Mascotte, CNRS/INRIA/UNSA, INRIA Sophia-Antipolis, 2004 route des Lucioles BP 93, 06902 Sophia-Antipolis Cedex, France fhavet@sophia.inria.fr

†LIRMM, 161 rue ADA, 34392 Montpellier cedex 5 France paul@lirmm.fr

Le problème SAT est NP-complet. Cependant, souvent, la résolution de ce problème sur des instances concrètes apparait bien plus facile qu'on ne pourrait l'espérer d'un problème NP-complet. Ainsi on se pose la question de savoir d'où la complexité de SAT provient. En particulier, comment le temps d'exécution des algorithmes dépend-il de certains paramètres et peut-on confiner l'inévitable explosion combinatoire sur ces paramètres? Considérons donc la complexité de SAT en fonction de quelques paramètres.

- **Taille de la clause.** Le nombre maximum k de litéraux dans une clause est un paramètre très naturel. Si $k = 2$ le problème est polynomial mais pour $k \geq 3$ le problème est NP-complet. Ainsi, ce paramètre ne semble pas utile pour capturer l'explosion combinatoire de SAT.

- **Nombre de variables.** Le nombre de variables influence de manière importante la complexité de SAT. En effet, s'il y a n variables, alors il y a au plus 2^n affectations de valeur de vérité. Ainsi le problème peut être résolu en 2^n étapes vérifiant pour chaque assignation que les m clauses sont vérifiées. Le problème se résout donc en $O(2^n \cdot m)$. Si les clauses ont une petite taille de meilleures bornes sont connues. Par exemple, si $k = 3$, 3-SAT peut être résolu en au plus $O(1,49^n \cdot m)$.

- **Nombre de clauses.** Si le nombre de clauses dans une formule est au plus m alors SAT peut être résolu en temps 1.24^m.

- **Longueur de la formule.** Si la longueur totale (le nombre total d'occurence de litéraux dans la formule) est bornée par l alors SAT peut être résolu en 1.08^l pas.

Dans tout ce papier, n dénotera la taille d'une instance, souvent le nombre de sommets d'un graphe. La complexité d'un problème sera toujours exprimée en fonction de n. Par simplicité, nous noterons souvent $n^{O(1)}$ une fonction quelconque bornée par un polynôme.

2.2 Deux problèmes d'optimisation

Soit Σ un alphabet fini. Un *problème de optimisation (NP)* sur Σ est une paire $O = (sol, cout)$ telle que:

(1) sol est une fonction définie sur Σ^* telle que:

  (a) il existe un polynôme $p \in \mathbf{N}[X]$ tel que si $y \in sol(x)$ alors $|y| \leq p(|x|)$.

  (b) Pour tout (x,y), on peut décider en temps polynomial si $y \in sol(x)$.

  Pour toute instance $x \in \Sigma^*$, les éléments de $sol(x)$ sont appelés *solutions* pour x.

(2) $cout$ est une fonction définie sur $\{(x,y) \mid x \in \Sigma^* \text{ et } y \in sol(x)\}$ à valeur dans $\mathbf{N}$ calculable en temps polynomial.

Si le problème d'optimisation est un problème de *minimisation* (resp. *maximisation*), la fonction opt_O sur Σ^* est définie par $opt_O(x) = \min\{cost(x,y) \mid y \in sol(x)\}$ (resp.$opt_O(x) = \max\{cost(x,y) \mid y \in sol(x)\}$). L'objectif d'un problème d'optimisation est de calculer le cout $opt_0(x)$. On représente le problème comme suit:
Instance: $x \in \Sigma^*$
Calculer: $opt_O(x)$.
Notons que calculer $opt_O(x)$ est souvent équivalent (à un facteur polynomial près) à trouver pour une instance x une solution $y \in sol(x)$ *optimale* c'est-à-dire telle que $cost(x,y) = opt_O(x)$.

2.2.1 Couverture minimum

Un premier exemple de problème de minimisation est celui de la couverture minimum d'un graphe. Une *couverture* d'un graphe $G = (V,E)$ est un ensemble de sommets $S \subset V$ tel que toute arête de E a au moins une de ses extrémités dans S. Une couverture est dite *minimum* si elle est de cardinalité minimum. La cardinalité d'une couverture minimum de G est dénotée $\nu(G)$.

Problème 2 (Couverture minimum)
Instance: Un graphe G.
Calculer: Taille minimum d'une couverture de G.

Une arête uv a deux extrémités et donc doit être couverte par au moins l'une d'elles. Ainsi toute couverture contient au moins un des deux sommets u et v. Cette observation simple garantit que l'algorithme récursif suivant renvoie une couverture minimale de G.

Algorithme 1 (VC(G))

1. Si G n'a pas d'arête rendre 0.
2. Sinon prendre une arête $e = uv$ de G.
3. Rendre $\min\{VC(G-u), VC(G-v)\} + 1$.

En fait, en faisant les appels récursifs de cet algorithme en parallèle, on construit un arbre de recherche binaire. Ainsi à chaque étape, nous avons 2^k couvertures partielles de cardinalité $k-1$. L'algorithme s'arrêtera dès qu'il aura trouvé une couverture qui sera alors minimum donc de taille $\nu(G)$. L'arbre de recherche sera donc de hauteur $2^{\nu(G)}$ (voir Figure 1). A chaque étape, correspondant à un nœud de l'arbre, on prend une arête et pour chacune des

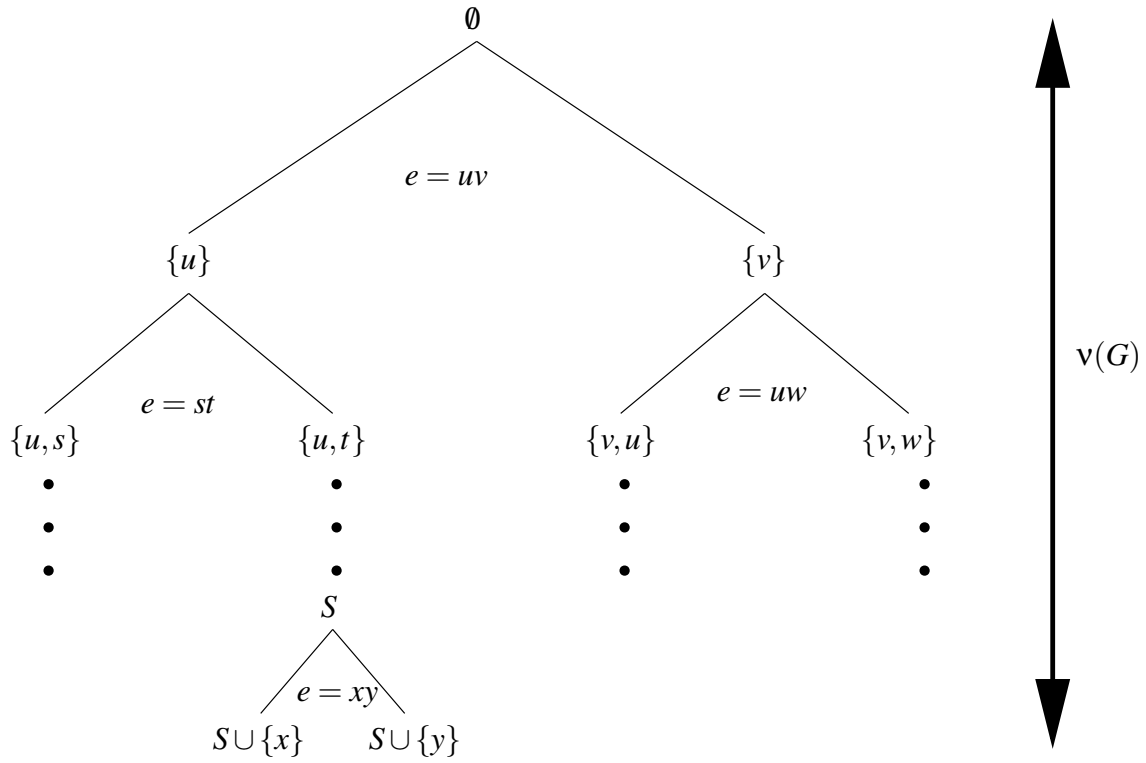

Figure 1: Arbre de recherche de la couverture minimale. Les noeuds sont indexés par les couvertures partielles

deux branches on enlève un sommet à G et on l'ajoute à S. Tout ceci se fait clairement en temps polynomial soit $n^{O(1)}$. Comme l'arbre de recherche a moins de $2^{\nu(G)+1}$ nœuds lorsque l'on tombe sur une couverture, la complexité est d'au plus $2^{\nu(G)+1} \cdot n^{O(1)}$. Ce type de complexité est intéressant car son exponentialité porte uniquement sur $\nu(G)$ et non sur la taille du graphe G.

2.2.2 Stable maximum

Un *stable* ou *ensemble indépendant* d'un graphe G est un ensemble $S \subset V$ de sommets deux à deux non-adjacents. On dénote par $\alpha(G)$ la cardinalité maximale d'un stable de G. Un problème classique de maximisation est celui du stable maximum.

Problème 3 (Stable maximum)
Instance: Un graphe G.
Calculer: Taille maximale d'un stable de G.

Un algorithme de force brute consiste à générer tous les ensembles stables en partant des singletons et en ajoutant un par un des sommets aux ensembles tant qu'ils restent stables. Cet algorithme fonctionne clairement en $O(n^{\alpha(G)})$. Malheureusement une telle complexité n'est pas satisfaisante car $\alpha(G)$ n'est pas une constante.

Le problème du stable maximum est intimement lié à celui de la couverture minimale.

Proposition 4 *Soit $G = (V,E)$ un graphe. $S \subset V$ est un stable si et seulement si $V \setminus S$ est une couverture. Donc $\alpha(G) = n - \nu(G)$.*

Preuve. Si S n'est pas un stable alors il y a deux sommets x et y dans S qui sont reliés par une arête. Ainsi $V \setminus S$ ne couvre pas xy et n'est donc pas une couverture. Réciproquement si S est un stable alors toutes les arêtes ont au moins une extrémité dans $V \setminus S$ donc S est une couverture. □

Ainsi on peut résoudre le problème du stable maximum en résolvant celui de la couverture, on obtient alors une complexité en $2^{n-\alpha(G)}.n^{O(1)}$. Celle-ci n'est alors pas non plus satisfaisante car l'exponentielle dépend de la taille du graphe.

Rappelons que l'idée de la complexité paramétrée est de considérer que si un paramètre est petit alors la dépendance du temps de calcul d'un algorithme en fonction de ce paramètre n'est pas trop importante. Le point essentiel est d'établir une ligne de démarcation entre les problèmes pouvant être résolus en temps de type $2^k.n$ d'un côté et n^k de l'autre, avec n la taille de l'instance et k celle du paramètre.

3 Problèmes paramétrés

3.1 Définitions

Comme en théorie classique de la complexité, les problèmes de décision sont décrits comme des languages sur des alphabets finis.

Soit Σ un alphabet fini.

Une *paramétrisation* de Σ^* est une fonction $\kappa : \Sigma^* \to \mathbb{N}$ calculable en temps polynomial. Un *problème paramétré* (sur Σ) est une paire (Q,κ) où $Q \subset \Sigma^*$ est un ensemble de mots sur Σ et κ une paramétrisation de Σ^*. Si (Q,k) est un problème paramétré sur l'alphabet Σ, on appelle *instance* de (Q,κ) un mot x de Σ^* et *paramètre* le nombre $\kappa(x)$ associé.

Habituellement, on représente un problème paramétré (Q,κ) sous la forme suivante:
Instance: $x \in \Sigma^*$.
Paramètre: $\kappa(x)$.
Question: Est-ce que $x \in Q$?

A tout problème d'optimisation O, on peut associer le *problème paramétré canonique* O_{can} suivant:
Instance: $x \in \Sigma^*$ et $k \in \mathbf{N}$.
Paramètre: k.
Question: Est-ce que $opt_O(x) \leq k$? si O est un problème de minimisation ou
Est-ce que $opt_O(x) \geq k$? si O est un problème de maximisation.

Ainsi le problème de la couverture minimum et du stable maximum se traduisent canoniquement de la façon suivante en un problème paramétré suivant:

Problème 5 (Couverture minimum (paramétré))
Instance: Un graphe G et un entier k.
Paramètre: k.
Question: G a-t-il une couverture de taille au plus k ? i.e. $\nu(G) \leq k$?

Problème 6 (Stable maximum (paramétré))
Entrée: Un graphe G et un entier k.
Paramètre: k.
Question: G a-t-il un stable de taille au moins k ? i.e. $\alpha(G) \geq k$?

Soit Σ un alphabet (fini) et $\kappa : \Sigma^* \to \mathbf{N}$ une paramétrisation. Un algorithme $\mathbf{A}$ est un algorithme *FPT selon* κ s'il existe une fonction $f : \mathbf{N} \to \mathbf{N}$ et un polynôme P tels que pour tout $x \in \Sigma^*$, le temps d'exécution de $\mathbf{A}$ sur l'entrée x est au plus

$$f(\kappa(x)) \cdot |x|^{O(1)}.$$

Un problème paramétré (Q, κ) est *FPT (fixed-parameter tractable)* ou *soluble à paramètre fixé* s'il existe un algorithme FPT selon κ qui reconnait Q.

Notons que la fonction f qui borne la dépendance dans le paramètre du temps d'exécution d'un algorithme FPT peut être arbitraire. Une critique est que c'est une liberté exagérée. En effet, si un temps d'exécution tel que $2^k \cdot n$ est plutôt bon pour de petites valeurs de k, (souvent même meilleur qu'un temps quadratique), un temps d'exécution de $2^{2^{2^{2^{2^k}}}} \cdot n$ est déjà rédhibitoire pour $k = 1$. La réponse à cette critique tient en deux points:

1) la classe des problèmes est robuste dans le sens où elle ne dépend pas d'un modèle de machines particulier, a des propriétés de clôture et permet une théorie mathématique.

2) Dans la grande majorité des cas, les problèmes "naturels" qui sont FPT ont une "faible" dépendance en le paramètre.

Remarque 7 Beaucoup de problèmes, comme celui de la couverture minimale ou du stable maximum, l'entrée est de la forme "$x \in \Sigma^*$ et un entier k" avec k le paramètre. Ceci revient à considérer $x' = xk \in \Sigma'^*$ avec $\Sigma' = \Sigma \cup \{0,1\}$. Ainsi l'instance x' et de taille $|x| + \log k$. Un simple calcul permet de montrer que le problème (Q, κ) est FPT si et seulement si Q est reconnaissable en temps $f(\kappa(x)) \cdot |x|^{O(1)}$.

Le problème de couverture minimale est FPT puisque l'Algorithme 1 a un temps d'exécution de $2^{\nu(G)+1} \cdot n^{O(1)}$.

Clairement, si un problème est polynomial alors pour toute paramétrisation κ, le problème paramétré (Q, κ) est FPT.

3.2 Importance de la paramétrisation

Le choix de la paramétrisation est important. Afin que le fait d'être FPT soit une information pertinente, il faut que le paramètre ne soit ni trop grand ni trop petit. En effet, si le paramètre $\kappa(x)$ est la taille de l'instance ou plus généralement s'il est borné inférieurement par une fonction de la taille de l'instance alors tout problème est FPT car le caractère exponentiel entre dans la fonction f. Par exemple, le problème SAT est FPT avec le paramètre n du

nombre de variables. Cependant, en général le nombre de variables augmente avec la taille de la formule normale conjonctive.

Proposition 8 *Soit $g : \mathbb{N} \to \mathbb{N}$ une fonction croissante non bornée, Σ un alphabet fini et $\kappa : \Sigma^* \to \mathbb{N}$ une paramétrisation telle que $\kappa(x) \geq g(|x|)$ pour toute instance $x \in \Sigma^*$.*

Alors pour tout ensemble $Q \subset \Sigma^*$ reconnaissable, le problème (Q, κ) est FPT.

Preuve. Soit $h : \mathbb{N} \to \mathbb{N}$ définie par $h(n) = \max\{m \in \mathbb{N} \mid g(m) \leq n\}$ si $n \geq g(1)$ et $h(n) = 1$ sinon. Comme g est croissante, h est bien définie et croissante. De plus, $h(g(n)) \geq n$ pour tout $n \in \mathbb{N}$. Ainsi pour tout $x \in \Sigma^*$ nous avons $h(\kappa(x)) \geq h(g(|x|)) \geq |x|$. Soit ϕ la fonction telle que toute instance $x \in Q$ est reconnaissable en temps $\phi(|x|)$. Alors $x \in Q$ est décidable en temps $f(\kappa(x)) = \phi(h(\kappa(x)))$. Donc (Q, κ) est FPT. □

A l'inverse si le paramètre est trop petit, il est également inutile. En effet, si κ est bornée alors pour toute fonction f, il existe une constante C_f telle que $f(\kappa(x)) \leq C_f$. Ainsi être FPT pour cette paramétrisation est équivalent à être décidable en temps polynomial sans paramètre.

Proposition 9 *Soit k un entier, Σ un alphabet fini et $\kappa : \Sigma^* \to \mathbb{N}$ une paramétrisation telle que $\kappa(x) \leq k$ pour tout instance $x \in \Sigma^*$. Alors pour tout $Q \subset \Sigma^*$, le problème (Q, κ) est FPT si et seulement si le problème Q est décidable en temps polynomial.*

3.3 Complexité paramétrée et complexité classique

3.3.1 NP-complétude et FPT

Un critère simple pour montrer qu'un problème n'est pas FPT est basé sur le fait que chaque niveau d'un problème FPT doit être soluble en temps polynomial.

Soit (Q, κ) un problème paramétré et $k \in \mathbb{N}$. Le $k$ème niveau de (Q, κ) est le problème classique: $(Q, \kappa)_k = \{x \in Q \mid \kappa(x) = k\}$. Par exemple, pour le problème de la couverture minimale, le $k$ème niveau est le problème de la k-couverture à k fixée.

Problème 10 (k-couverture)
Entrée: Un graphe G.
Question: G a-t-il une couverture de taille k ?

Considérons par exemple le problème de la coloration de graphe. Rappelons qu'un graphe G est *k-colorable* s'il admet une coloration propre en k couleurs, i.e. une application $c : V(G) \to \{1, \ldots, k\}$ telle que pour toute arête $uv \in E(G)$, $c(u) \neq c(v)$. Si on paramétrise le problème de la coloration avec le nombre de couleurs, on obtient le problème suivant:

Problème 11 (Colorabilité (paramétré))
Entrée: Un graphe G et un entier k.
Paramètre: k.
Question: G admet-il une k-coloration?

Le troisième niveau de ce problème est le problème classique de la 3-colorabilité qui est NP-complet. Ainsi le problème de la colorabilité n'est pas FPT.

Malheureusement, pour beaucoup de problèmes paramétrés qui sont pressentis pour ne pas être FPT, il n'existe pas de réduction aussi simple à la théorie de la NP-complétude. Par exemple, il est fortement pressenti que le problème du stable maximum n'est pas FPT mais chaque niveau est soluble en temps polynomial.

3.3.2 Approximabilité et FPT

Soit $O = (sol, cout)$ un problème d'optimisation.

Pour toute instance x de O et toute solution $y \in sol(x)$ pour x, le *rapport d'approximation* $r(x,y)$ de y est défini par $r(x,y) = \dfrac{cout(x,y)}{opt_O(x,y}$ si O est un problème de minimisation et $r(x,y) = \dfrac{opt_O(x,y}{cout(x,y)}$ si O est un problème de maximisation. Notons que le rapport d'approximation est toujours un nombre supérieur ou égal à 1 et que plus de rapport est proche de 1 meilleure est la solution.

Soit $\varepsilon > 0$ un réel. Un algorithme ε-*approché* pour O est un algorithme qui pour toute instance x calcule une solution y telle que $r(x,y) \leq (1+\varepsilon)$. Un *schéma d'approximation polynomial (PTAS=polynomial time approximation scheme)* pour un problème O est un algorithme qui prend en entrée une pair $(x,k) \in \Sigma^* \times \mathbb{N}$ et qui pour tout entier k fixé, l'algorithme est $(1/k)$-approché en temps polynomial.

La plupart des schémas d'approximation polynomiaux ont un temps d'exécution de la forme $n^{\Omega(k)}$, ce qui implique que pour des approximations suffisamment bonnes, le temps d'exécution devient rapidement trop long quand la taille des instances augmente. Un PTAS est un *schéma d'approximation fortement polynomial (FPTAS= fully polynomial time approximation scheme)* si son temps d'exécution est polynomial en $|x|+k$. Malheureusement, très peu de problèmes d'optimisation ont un FPTAS. Cependant, si la précision requise de l'approximation n'est pas trop importante, on se retrouve avec un paramètre k qui est petit. Par exemple $k = 10$ pour une approximation à 10%. En paramétrant les schémas d'approximation par ce paramètre d'erreur k, on obtient la notion intermédiaire suivante: un PTAS est un *schéma d'approximation polynomial efficace (EPTAS = efficient polynomial time approximation scheme)* s'il existe une fonction f quelconque et un polynôme p tel que le temps d'exécution sur l'entrée $(x,k) \in \Sigma^* \times \mathbb{N}$ est au plus $f(k) \cdot p(|x|)$. En d'autres termes, un EPTAS est un algorithme FPT suivant la paramétrisation $(x,k) \mapsto k$ de $\Sigma^* \times \mathbb{N}$.

Clairement un FPTAS est un EPTAS et un EPTAS est un PTAS. La notion d'EPTAS semble être une notion intermédiaire raisonnable entre celle de PTAS et la notion très contraignante de FPTAS. Un exemple connu d'EPTAS est le schéma d'approximation d'Arora [2] pour le problème du voyageur de commerce euclidien.

Le résultat suivant établit un lien entre l'existance d'un EPTAS pour un problème d'optimisation et le fait qu'avec la paramétrisation canonique le problème soit FPT. Ce résultat est simple mais intéressant car il relie deux problèmes paramétrés complétement différent issu du même problème d'optimisation.

Théorème 12 *Si un problème d'optimisation O possède un EPTAS alors le problème paramétré O_{can} dérivant canoniquement de O est FPT.*

Preuve. Soit $O = (sol, cout)$ un problème de minimisation sur l'alphabet Σ. (La preuve pour un problème de maximisation est identique.) Soit $\mathbf{A}$ un EPTAS pour O de temps d'exécution $f(k) \cdot |X|^{O(1)}$ pour une certaine fonction f.

Considérons l'algorithme $\mathbf{A}_{can}$ qui étant donné une instance (x,k) de O_{can} calcule la solution y de $\mathbf{A}$ sur l'instance $(x,k+1)$. Il accepte l'instance si $cout(x,y) \leq k$ et rejette sinon. Clairement $\mathbf{A}_{can}$ est un algorithme FPT car $\mathbf{A}$ l'est. Pour voir que $\mathbf{A}_{can}$ est correct, distinguons deux cas: si $cout(x,y) \leq k$ alors $opt_O(x) \leq k$ et donc (x,k) doit être accepté. Si $cout(x,y) \geq k+1$ alors $opt_O(x) = \dfrac{cout(x,y)}{r(x,y)} \geq \dfrac{k+1}{1+\frac{1}{k+1}} > k$. Ainsi l'instance (x,k) doit être rejetée. □

Au vu de ce théorème, on peut montrer la non-existence d'EPTAS en établissant qu'un problème paramétré n'est pas FPT. Notons que la réciproque du Théorème 12 est fausse. Il est connu que le problème de la couverture minimum n'admet pas de PTAS (sauf si P=NP) mais nous avons vu qu'avec la paramétrisation canonique il est FPT.

4 Noyaux

Soit (Q,κ) un problème paramétré sur Σ. Une fonction calculable en temps polynomial $K : \Sigma^* \to \Sigma^*$ est une *nucléarisation (kernelisation)* de (Q,κ) s'il existe une fonction $h : \mathbb{N} \to \mathbb{N}$ tel que pour toute instance $x \in \Sigma^*$ nous avons

$$(x \in Q \Leftrightarrow K(x) \in Q) \text{ et } |K(x)| \leq h(\kappa(x)).$$

De manière informelle, une *nucléarisation* est une réduction (surjective) d'un problème dans lui-même telle que le paramètre de l'image est borné.

Une nucléarisation est intéressante car si un problème paramétré $(Q.\kappa)$ admet une nucléarisation K alors il est FPT. En effet, considérons l'algorithme suivant : pour toute instance $x \in \Sigma^*$, calculons $K(x)$ (cela se fait en temps polynomial) et utilisons un algorithme (de complexité quelconque) qui reconnait si $K(x) \in Q$. Comme $|K(x)| \leq h(\kappa(x))$, cette dernière étape prend un temps borné en focntion de $\kappa(x)$.

Une nucléarisation est souvent une succession de règles de réduction. Illustrons cela sur l'exemple de la couverture minimum. Pour cela, nous avons besoin des observations suivantes:

Proposition 13 *Soit G un graphe et x un sommet de G.*

(i) Si x est un sommet isolé alors il n'est dans aucune couverture minimum.

(ii) Si x a un unique voisin y, alors il existe une couverture minimum comprenant y et pas x.

(iii) Si x est de degré supérieur à k alors toute couverture de taille au plus k contient x.

Preuve.

(i) Considérons une couverture S qui contient x alors $S \setminus \{x\}$ est également une couverture. En effet, toute arête a une extrémité dans S et donc dans $S \setminus \{x\}$ car x n'est incident à aucune arête. S n'est donc pas une couverture minimum.

(ii) Soit S une couverture contenant x. Posons $S' = (S \setminus \{x\}) \cup \{y\}$. Alors S' est également une couverture car l'arête xy est couverte par y et toutes les autres ont une extrémité dans S et donc dans $S \setminus \{x\}$. De plus $|S'| \leq |S|$.

(iii) Si une couverture ne contient pas un sommet v alors elle doit contenir tous ses voisins pour couvrir les arêtes incidentes à v. Ainsi si $d(x) > k$ et S est une couverture ne contenant pas x, $|S| \geq d(x) > k$.

□

Considérons la fonction VC-K définit inductivement par l'algorithme suivant:

Algorithme 2 (VC-K(G,k))

1. Si G a un sommet isolé x alors VC-K(G,k) = VC-K$(G-x,k)$.
2. Si G a un sommet x ayant un unique voisin y alors VC-K(G,k) = VC-K$(G-\{x,y\},k-1)$.
3. Si G a un sommet x de degré supérieur à k alors VC-K(G,k) = VC-K$(G-x,k-1)$.
4. Sinon VC-K(G,k)=(G,k)

La Proposition 13 garantit que VC-K(G,k) est une instance positive du problème de couverture minimum si et seulement si (G,k) l'est également. De plus, l'algorithme ci-dessus est clairement polynomial. Donc, pour que VC-K soit une nucléarisation du problème de couverture minimum alors il faut vérifier qu'il existe une fonction h telle que pour toute instance (G,k), $|\text{VC-K}(G,k)| \leq h(k)$. Pour cela, notons $(G_K,k_K) = \text{VC-K}(G,k)$. Il est clair que $k_K \leq k$, il suffit donc de montrer que la taille de G_K est bornée en fonction de k.

Lemme 14 (Buss) *Si G possède une couverture de taille au plus k alors G_k a au plus k^2+k sommets et au plus k^2 arêtes.*

Preuve. Remarquons qu'à cause de l'étape de réduction 3 de l'Algorithme 2, tous les sommets de G_K ont degré au plus k. Si G admet une couvertuse de taille au plus k alors G_K admet une couverture S est une couverture de taille au plus $k_K \leq k$. Maintenant toutes les arêtes de G_K sont incidentes aux k_K sommets de S. Le graphe G_K a donc au plus $k \times k_K \leq k^2$ arêtes. De plus chaque sommet de G_K est incident à une arête à cause de l'étape de réduction 1 de l'Algorithme 2 et donc dans S ou adjacent à un sommet de S. Donc $|V(G_K)| = |S \cup N(S)| \leq k+k^2$. □

Nous avons vu qu'une condition suffisante pour qu'un problème soit FPT est l'existence d'une nucléarisation. En fait, cette condition est également nécessaire.

Théorème 15 (Niedermeier) *Soit (Q,κ) un problème paramétré. Les assertions suivantes sont équivalentes:*

(i) *(Q,κ) est FPT;*

(ii) *Q est reconnaissable et (Q,κ) admet une nucléarisation.*

Preuve.

Il nous faut prouver $(i) \rightarrow (ii)$.

Soient Σ l'alphabet de (Q,κ) et $\mathbf{A}$ un algorithme résolvant (Q,κ) en temps $f(\kappa(x)) \cdots p(|x|)$ avec f une focntion quelconque et p un polynôme. Sans perte de génértalité, on peut supposer que $p(n) \geq n$ pour tout $n \in \mathbf{N}$. Si $Q = \emptyset$ ou $Q = \Sigma^*$ alors (Q,k) admet la nucléarisation triviale qui envoie toute instance $x \in \Sigma^*$ sur le mot vide. On peut donc supposer qu'il existe $x_0 \in Q$ et $x_1 \in \Sigma^* \setminus Q$.

L'algorithme $\mathbf{A}'$ suivant calcule une nucléarisation K pour (Q,κ). Pour toute instance $x \in \Sigma^*$ telle que $|X| = n$ et $\kappa(x) = k$, l'algorithme $\mathbf{A}'$ simule $p(n) \cdot p(n)$ pas de $\mathbf{A}$. Si $\mathbf{A}$ s'arrête et accepte (resp. rejette) x alors $\mathbf{A}'$ renvoie $K(x) = x_0$ (resp. $K(x) = x_1$). Si $\mathbf{A}$ ne s'arrête pas en au plus $p(n) \cdot p(n)$ pas alors $\mathbf{A}'$ renvoie $K(x) = x$. Notons que dans ce cas $n \leq p(n) \leq f(k)$. Donc $|K(x))| \leq \max\{|x_0|,|x_1|,f(\kappa(x))\} = h(\kappa(x))$. De plus, $\mathbf{A}'$ calcule $K(x)$ en temps polynomial et $(x \in Q \Leftrightarrow K(x) \in Q)$. Ainsi K est donc bien une nucléarisation. □

Au vu de ce résultat, on pourrait se dire que le but de la complexité paramétrée est de montrer l'existence d'un noyau. Cependant, n'oublions pas que le but recherché par l'algorithmique paramétrée est d'une part d'obtenir des algorithmes efficaces pratiquement et d'autre part de mieux classifier les problèmes suivant leur complexité. Ainsi la preuve de l'existence d'un noyau pour un problème paramtré n'est pas la fin de l'histoire. En effet, si le problème à un noyau important (disons exponentiel en le paramètre) alors l'explosion combinatoire de l'algorithme associé à ce noyau sera énorme. En revanche, si le noyau est petit (disons polynomial en le paramètre) alors l'explosion combinatoire sera moindre et l'algorithme a une chance d'être efficace. Ainsi on distingue les problèmes a ayant des noyaux polynomiaux de ceux qui ne peuvent en avoir. Voir [3].

5 Quelques techniques en FPT

Outre la réduction à un noyau, plusieurs techniques génériques ont été développées pour montrer que des problèmes sont FPT.

- Arbres de recherche bornés: L'idée est d'effectuer une recherche systématique de l'espace (énorme) des solutions. Pour cela, on suit un arbre de recherche qui sera borné de la manière suivante: en temps polynomial, on trouve un "petit ensemble" tel qu'au moins un de ces éléments fasse partie d'une solution optimale au problème. Par exemple, dans le cas de la couverture minimum, le "petit ensemble" est l'ensemble de deux extrémités d'une même arête. Cela a mené à l'arbre de recherche de taille 2^k du paragraphe 2.2.1. Cette méthode a été utilisée pour de nombreux problèmes comme le stable maximum ou le dominant minimum dans les graphes planaires. Nous renvoyons le lecteur au Chapitre 3 de [11].

- Color Coding: Cette technique a été introduite par Alon, Yuster et Zwick [1]. Elle est basée sur des techniques de dérandomisation et de hashage. Elle a été appliquée avec succès aux problèmes de recherches de petits sous-graphes.

- Compression itérative: Cette méthode a été introduite par Reed, Smith et Vetta [12]. L'idée générale est de calculer une solution pour une instance du problème en utilisant les informations données par une solution d'une instance plus petite. Typiquement, on a un problème de minimisation avec comme paramètre la taille de la solution, pour lequel si S est une solution pour $G-v$ alors $S\cup\{v\}$ est également solution pour G. Ainsi à partir d'une solution de taille k pour $G-v$ on en désuit une solution de taille $k+1$ pour G. Reste à faire une pahse de compression qui à partir d'une telle solution en calcule une de taille k (si elle existe). Cette méthode a été utilisée pour différents problèmes comme le transverse minimum des cycles impair au sens des sommets [12] et au sens des arêtes [9], le "feedback vertex set" dans les graphes [9] et les tournois [5], le nombre minimum de sommets dont la suppression rend un graphe chordal. [10].

- Largeur d'arborescence: De nombreux résultats en complexité repose sur la théorie des décompostion de type arborescente des graphes et de leurs paramètres de largeur associés (largeur d'arborescence, largeur de branche, ...). Par exemple, un important résultat de Courcelle [4] montre que toute propriété expressible en *logique du second ordre monadique* peut être testée en temps FPT avec la largeur d'arborescence en paramètre. Malheureusement, la fonction de ce paramètre est une énorme tour d'exponentiel (voir [8]) et les algorithmes sont inefficaces en pratique.

References

[1] N. Alon, R. Yuster and U. Zwick. Color coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *Proc. Symp. Theory of Computing (STOC)*, 326–335, 1994.

[2] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[3] H. L. Bodlaender, R. G. Downey, M. R. Fellows and D. Hemelin. On problems without polynomial kernels. *Technical Report UU-CS-2007-046*, Utrecht University, The Netherlands.

[4] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In *Graph Structure Theory 1991, Contemporary Mathematics* 147:565–590, 1993.

[5] M. Dom, J. Guo, F. uffner, R. Niedermeier, and A. Tru. Fixed-Parameter Tractability. Results for Feedback Set Problems in Tournaments. In *Proc. 6th CIAC-06, Lecture Notes in Comput. Sci.* 3998: 320–331, 2006.

[6] R. Downey et M. Fellows. *Parameterized Complexity*, Springer, 1999.

[7] J. Flum et M. Grohe. *Parameterized Complexity Theory*, Springer, 2006.

[8] M. Frick and M. Grohe. The Complexity of First-Order and Monadic Second-Order Logic Revisited. In *LICS 2002*: 215–224, 2002.

[9] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier and S. Wernicke. Improved fixed-parameter algorithms for two feedback set problems. In *Proc. of WADS 2005, Lecture Notes in Comput. Sci.* 3608: 158–168, 2005.

[10] Daniel Marx. Chordal Deletion Is Fixed-Parameter Tractable. In *Proc. of WG 2006, Lecture Notes in Comput. Sci.* 4271: 37–48, 2006.

[11] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.

[12] B. Reed, K. Smith and A. Vetta. Finding odd cycle transversals. *Operations Research Letters* 32:299-301, 2004.