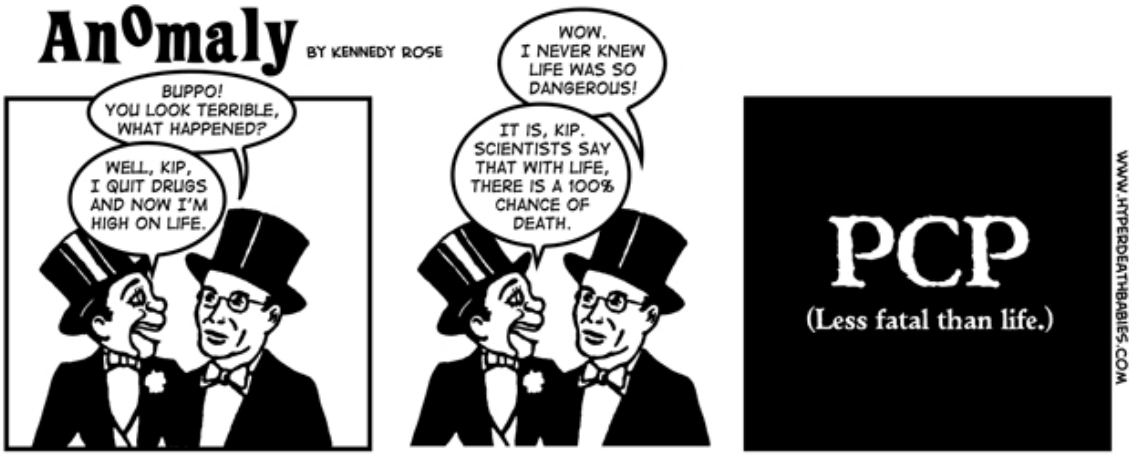
Gadgets, approximabilité et PCP

Michaël Monerau

Mai 2009



Introduction

Un des grands mystères de la théorie de la complexité reste de trouver les réductions de problèmes à d'autres. Après quelques dizaines d'années de recherche dans ce domaine, on connaît maintenant un grand nombre de telle réductions, qui permettent de n'étudier que quelques problèmes pour en déduire des résultats sur une grande famille d'autres.

Dans [TSSW00], les auteurs présentent une méthode algorithmique pour trouver de telles réductions, qui présente l'avantage, en plus d'automatiser des astuces parfois difficiles à trouver, de donner des preuves d'optimalité dans certains cas.

Dans ce travail, nous nous attacherons à comprendre ce cheminement. Essentiellement, ce qui suit est une redite un petit peu enrobée de [TSSW00], avec quelques exemples et explications supplémentaires pour mieux saisir les notions.

Dans un premier temps, nous introduisons les notions et définitions indispensables à l'énoncé du résultat principal. Ensuite, nous étudions comment utiliser cette méthode algorithmique sur trois exemples : MAXCUT, MAX-2CSP et MAX 3ConjSAT.

Chacun de ces trois problèmes d'optimisation est relié à la classe **PCP**, et donne donc des informations sur celle-ci. Cependant, nous ne faisons que les énoncer car leur preuve sort du cadre de cet exposé.

Plan

1 Gadgets, témoins et approximation

1.1 Contraintes, clauses, gadgets

1.1.1 Famille de contraintes

Dans toute la suite, on considère des fonctions booléennes d'arité variable. On notera $f^{(k)}$ une fonction booléenne $f : \{0,1\}^k \longrightarrow \{0,1\}$ d'arité k lorsque cela améliore la lisibilité.

Ces fonctions seront principalement vues comme des *contraintes* sur les variables auxquelles elles sont appliquées. Plus précisément, $f(X_1, \ldots, X_k)$ sera vue comme la contrainte sur $(X_1, \ldots, X_k)$: $f(X_1, \ldots, X_k) = 1$.

[1.A] Définition ***(Famille de contraintes)***

On appelle famille de contraintes *un ensemble $\mathcal{F}$ de fonctions de contraintes, sans restriction sur l'arité de ces fonctions.*

Donnons d'ores et déjà les familles de contraintes qu'on utilisera par la suite :

- **CUT** : c'est le singleton composé de la contrainte $CUT(a, b) = a \oplus b$.
- **CUT/0** : c'est l'ensemble $\{CUT, T_0\}$, où $T_0(a) = 0 \oplus a = a$.
- **CUT/1** : c'est l'ensemble $\{CUT, T_1\}$, où $T_0(a) = 1 \oplus a = \neg a$.
- **Parity Check (PC)** : c'est l'ensemble $\{PC_0, PC_1\}$ où PC_i est définie pour $i \in \{0,1\}$ par :

$$PC_i(a,b,c) = \begin{cases} 1 & \text{si } a \oplus b \oplus c = i \\ 0 & \text{sinon} \end{cases}$$

  où $\oplus$ est le XOR. PC_i renvoie donc si parmi a, b et c il y a un nombre pair ($i = 0$) ou impair ($i = 1$) de 1.

- **Binary Constraint Satisfaction Problem (2CSP)** : c'est l'ensemble des 16 fonctions binaires :

$$\textbf{2CSP} = \{f : \{0,1\}^2 \longrightarrow \{0,1\}\}$$

- **3-Conjonctive SAT (3ConjSAT)** : c'est l'ensemble $\{f_{000}, f_{001}, f_{011}, f_{111}\}$ où :
  - $f_{000}(a,b,c) = a \wedge b \wedge c$
  - $f_{001}(a,b,c) = a \wedge b \wedge \neg c$
  - $f_{011}(a,b,c) = a \wedge \neg b \wedge \neg c$
  - $f_{111}(a,b,c) = \neg a \wedge \neg b \wedge \neg c$

1.1.2 Clauses

Afin de formaliser la notion de problème, on pose la :

[1.B] Définition ***(Clause)***

Une ***clause*** *C sur une famille de contraintes $\mathcal{F}$ à propos de variables booléennes $(X_1, \ldots, X_n)$ est une paire $C = (f^{(k)}, (i_1, \ldots, i_k))$ où $f \in \mathcal{F}$ et $(i_1, \ldots, i_k)$ est une famille d'entiers distincts de $[\![1, n]\!]$.*

La clause est dite ***satisfaite*** *par une assignation $\vec{a} = (a_1, \ldots, a_n)$ si :*

$$C(\vec{a}) \overset{déf}{=} f\left(a_{i_1}, \ldots, a_{i_k}\right) \quad = \quad 1$$

Ce formalisme permet d'exprimer simplement des problèmes bien connus de manière unifiée. Par exemple, une instance du problème **SAT** peut être vue comme un ensemble de clauses (ou bien une seule clause) où les contraintes f sont issues de la famille **SAT**, qui n'est autre que l'ensemble de toutes

les fonctions booléennes, d'arité quelconque. Pour parler de **3SAT**, il suffit de se restreindre pour $\mathcal{F}$ à l'ensemble des fonctions disjonctives d'arité 3.

Cela permet de parler des problèmes d'optimisation :

[1.C] Définition ***(Problème d'optimisation MAX-$\mathcal{F}$)***

Pour une famille de contraintes $\mathcal{F}$ finie*, on appelle* ***MAX-$\mathcal{F}$*** *le problème d'optimisation qui suit :*

Entrée *: un ensemble de m clauses sur $\mathcal{F}$: $(C_1, \ldots, C_m)$ à propos de n variables booléennes $(X_1, \ldots, X_n)$, ainsi qu'un ensemble de m poids positifs $(w_1, \ldots, w_m)$. Une instance est donc un triplet $\left(\vec{X}, \vec{C}, \vec{w}\right)$.*

Sortie *: une assignation $\vec{a}$ pour $\vec{X}$ qui maximise la quantité $\sum_{j=1}^{m} w_j\, C_j(\vec{a})$, ie. le poids des clauses satisfaites.*

À l'aide de cette définition, on retrouve des problèmes d'optimisation connus comme **MAXSAT**, **MAX3SAT**, **MAXCUT**, etc.

Par exemple pour **MAXCUT**. On prend un graphe $G = (V, E)$. On introduit une variable booléenne par sommet $i \in [\![1, |V|]\!]$, notée X_i qui dit à quelle sous-partie de la coupe le sommet appartient. Ensuite, on introduit une clause par arête (i, j) : $C_{i,j} = (CUT, (i, j))$. Enfin, on place les poids w_j des arêtes. Alors la définition qu'on a donnée ici de **MAXCUT** sur cette instance correspond à la version connue définie à partir du graphe G.

On remarque qu'on travaille ici avec les versions pondérées des problèmes d'approximation. En effet, ils ne sont pas plus difficiles à approximer que les versions non pondérées (pour les problèmes qu'on considère ici) comme justifié dans [CST96] ; mais ils sont plus souples à manipuler.

1.1.3 Gadgets

1.1.3.1 α-gadgets

Toute la philosophie du travail qui suit repose sur la réduction de problèmes de satisfiabilité à des problèmes d'optimisation. Le théorème principal permet de trouver de telles réductions de manière automatique (et optimale parfois). La notion de réduction doit cependant être précisée. C'est pourquoi on introduit la notion de :

[1.D] Définition *(α-gadget)*

Soient $\alpha \geq 1$ un réel, $f^{(k)}$ une fonction booléenne et une famille de contraintes $\mathcal{F}$.

Un α-gadget réduisant f à $\mathcal{F}$ est un ensemble de variables booléennes secondaires $(Y_1, \ldots, Y_n)$, une famille finie de poids positifs $(w_1, \ldots, w_m)$ et des contraintes associées $(C_1, \ldots, C_m)$ sur $\mathcal{F}$ à propos de variables principales *$(X_1, \ldots, X_k)$, tels que :*

Pour toute assignation $\vec{a}$ de $(X_1, \ldots, X_k)$ et pour toute assignation $\vec{b}$ de $(Y_1, \ldots, Y_n)$, les conditions suivantes sont respectées :

$$(\forall \vec{a},\ f(\vec{a}) = 1)\ (\forall \vec{b}), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad \leq \quad \alpha \tag{1}$$

$$(\forall \vec{a},\ f(\vec{a}) = 1)\ (\exists \vec{b}), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad = \quad \alpha \tag{2}$$

$$(\forall \vec{a},\ f(\vec{a}) = 0)\ (\forall \vec{b}), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad \leq \quad \alpha - 1 \tag{3}$$

*Le gadget est **strict** s'il respecte de plus :*

$$(\forall \vec{a},\ f(\vec{a}) = 0)\ (\exists \vec{b}), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad = \quad \alpha - 1 \tag{4}$$

On notera un tel gadget : $\Gamma = (\vec{Y}, \vec{C}, \vec{w})$.

L'idée sous-jacente est, comme annoncé plus haut, de réduire le problème de la satisfiabilité de f à certaine instance du problème **MAX**-$\mathcal{F}$ grâce à l'introduction de variables supplémentaires (les variables *secondaires*).

Un des intérêts conceptuels de l'introduction de ces gadgets est le théorème suivant :

[1.E] Théorème

Soit une famille de contraintes $\mathcal{F}$. On suppose qu'il existe un α_0-gadget qui réduit PC_0 à $\mathcal{F}$ et un α_1-gadget qui réduit PC_1 à $\mathcal{F}$.

*Alors pour tout $\varepsilon > 0$, **MAX**-$\mathcal{F}$ est difficile à approximer dans un facteur $1 - \frac{1}{\alpha_0 + \alpha_1} + \varepsilon$.*

Preuve de [1.E].
Elle reste introuvable jusqu'à ce jour. [TSSW00] dit qu'elle se trouve dans [Has97], et [Has97] lui renvoie la balle (au moins dans la version récente de [Has97] disponible sur la homepage de l'auteur). □

D'autre part, on connaît des liens entre l'approximabilité de certains problèmes de type MAX-$\mathcal{F}$ et la classe $\mathbf{PCP}_{c,s}$. L'existence de gadgets, associée à ce théorème, peut donc donner de précieuses informations sur le comportement des classes $\mathbf{PCP}_{c,s}$ et nous allons voir que c'est effectivement le cas.

On remarque que si on dispose d'un α-gadget, alors on peut construire un α'-gadget simplement en multipliant tous les poids par α'/α. C'est cohérent avec le fait que dans le théorème [1.E], plus la valeur de α est faible, plus le résultat d'inapproximabilité est fort (le problème est difficile à approximer même avec un faible taux de précision). À la limite, la valeur $\alpha = 1$ est une réduction optimale de f vers **MAX**-$\mathcal{F}$.

1.1.3.2 (α, S)-gadgets

Il s'avèrera techniquement utile de raffiner la notion de gadget :

[1.F] Définition *((α, S)-gadget)*

On se place dans le même cadre que dans la définition [1.D], mais on dispose en plus d'une partie $S \subseteq \{0,1\}^k$.

Un (α, S)-gadget réduisant f à $\mathcal{F}$ vérifie les équations :

$$(\forall \vec{a} \in \{0,1\}^k,\ f(\vec{a}) = 1)\ (\forall \vec{b} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad \leq \quad \alpha \tag{5}$$

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 1)\ (\exists \vec{b} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad = \quad \alpha \tag{6}$$

$$(\forall \vec{a} \in \{0,1\}^k,\ f(\vec{a}) = 0)\ (\forall \vec{b} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad \leq \quad \alpha - 1 \tag{7}$$

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 0)\ (\exists \vec{b} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b}) \quad = \quad \alpha - 1 \tag{8}$$

Remarquons qu'on n'a pas ici de notion de (α, S)-gadget strict. La proposition qui suit est un fait découlant immédiatement des définitions :

[1.G] Proposition *(α-gadgets et (α, S)-gadgets)*

Soit une contrainte $f^{(k)}$, et soient $S_1 = \{\vec{a} \in \{0,1\}^k \mid f(\vec{a}) = 1\}$ et $S_2 = \{0,1\}^k$. Alors :

- *Un (α, S_1)-gadget réduisant f à $\mathcal{F}$ est un α-gadget réduisant f à $\mathcal{F}$.*
- *Un (α, S_2)-gadget réduisant f à $\mathcal{F}$ est un α-gadget strict réduisant f à $\mathcal{F}$.*

1.1.3.3 Exemples de gadgets

Voyons quelques exemples pour voir à quoi correspond cette notion de gadget.

Construisons une réduction d'une fonction de **3SAT** vers **2SAT**. Si $f(X_1, X_2, X_3) = X_1 \vee X_2 \vee X_3$, alors on introduit une variable secondaire Y et on pose les clauses :

$$X_1, \quad X_2, \quad X_3, \quad \neg X_1 \vee \neg X_2, \quad \neg X_2 \vee \neg X_3, \quad \neg X_3 \vee \neg X_1,$$

$$Y, \quad X_1 \vee \neg Y, \quad X_2 \vee \neg Y, \quad X_3 \vee \neg Y$$

On peut alors vérifier que pour toute assignation des variables principales $\vec{X}$, si $f(\vec{X}) = 1$, alors 7 des 10 clauses sont satisfiables, et sinon au plus 6 le sont. On construit donc le 7-gadget correspondant en attribuant simplement le poids 1 à chaque clause.

On a donc réduit la résolution d'une fonction de **3SAT** à la résolution du problème MAX-2SAT. Plus généralement, cela permet une réduction de tout problème 3SAT à MAX-2SAT (il suffit de faire cette manipulation sur chaque clause).

1.2 Familles héréditaires et complémentation-closes, matrices témoins

1.2.1 Propriétés particulières de familles de contraintes

Dans la suite, nous allons déduire des résultats généraux sur certaines familles de contraintes simplement parce qu'elles présentent ou non les deux propriétés suivantes :

[1.H] Définition ***(Famille héréditaire)***

Une famille de contraintes $\mathcal{F}$ est dite ***héréditaire*** *si pour toute $f^{(k)} \in \mathcal{F}$, pour tous indices $i, j \in [\![1, n]\!]$, la fonction f restreinte à $X_i \equiv X_j$ et vue comme une fonction d'arité $k-1$, est égale à une $g^{(k-1)} \in \mathcal{F} \cup \{\mathbf{0}, \mathbf{1}\}$ (à l'ordre des arguments près), où $\mathbf{0}$ et $\mathbf{1}$ désignent les fonctions constantes.*

[1.I] Définition ***(Famille complémentation-close)***

Une famille de contraintes $\mathcal{F}$ est dite ***complémentation-close*** *si elle est héréditaire et si, de plus, pour toute $f^{(k)} \in \mathcal{F}$ et pour tout indice $i \in [\![1, k]\!]$, la fonction f' est contenue dans $\mathcal{F}$ où f' est définie par $f'(X_1, \ldots, X_k) = f(X_1, \ldots, X_{i-1}, \neg X_i, X_{i+1}, \ldots, X_k)$.*

1.2.2 Matrices témoins

On parlera d'un témoin pour un (α, S)-gadget pour exprimer le fait qu'on exhibe une solution aux problèmes d'existence posés par la définition [1.F]. Formellement :

[1.J] Définition ***(Témoin)***

Soit Γ un (α, S)-gadget réduisant une fonction $f^{(k)}$ à une famille $\mathcal{F}$.

On dit que la fonction $b : S \longrightarrow \{0,1\}^n$ est un ***témoin*** *pour Γ si pour tout $\vec{a} \in \{0,1\}^k$, $b(\vec{a})$ satisfait aux équations (6) et (8), c'est-à-dire que :*

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 1) \quad \sum_{j=1}^{m} w_j\, C_j\,(\vec{a}, b(\vec{a})) \quad = \quad \alpha \tag{9}$$

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 0) \quad \sum_{j=1}^{m} w_j\, C_j\,(\vec{a}, b(\vec{a})) \quad = \quad \alpha - 1 \tag{10}$$

On organise les témoins sous forme de **matrices témoins** de taille $|S| \times (k+n)$ où les lignes sont les vecteurs $(\vec{a}, b(\vec{a}))$ pour $\vec{a}$ décrivant S. Une telle matrice est donc formée uniquement de 0 et de 1. Graphiquement, en décrivant l'ensemble S par l'énumération $S = \{\vec{a}^{(1)}, \ldots, \vec{a}^{(|S|)}\}$, une matrice témoin est de la forme :

$$\begin{array}{c} \\ (\vec{a}^{(1)}, b(\vec{a}^{(1)})) \\ \\ \\ \\ \\ (\vec{a}^{(|S|)}, b(\vec{a}^{(|S|)})) \end{array} \begin{array}{c} \begin{array}{cccccccc} X_1 & \ldots & X_k & Y_1 & \ldots & \ldots & Y_n \end{array} \\ \begin{pmatrix} \vec{a}_1^{(1)} & \ldots & \vec{a}_k^{(1)} & b(\vec{a}^{(1)})_1 & \ldots & \ldots & b(\vec{a}^{(1)})_n \\ \vdots & & \vdots & \vdots & & & \vdots \\ \vdots & & \vdots & \vdots & & & \vdots \\ \vdots & & \vdots & \vdots & & & \vdots \\ \vec{a}_1^{(|S|)} & \ldots & \vec{a}_k^{(|S|)} & b(\vec{a}^{(|S|)})_1 & \ldots & \ldots & b(\vec{a}^{(|S|)})_n \end{pmatrix} \end{array}$$

Et on remarque que les colonnes sont donc les différentes assignations aux variables, tant principales que secondaires.

Pour finir cette série de définitions, on définit des matrices "complètes" au sens où elles contiennent toutes les matrices témoins. En effet, les matrices témoins n'étant formées que de 0 et de 1, à nombre de lignes fixe (si on suppose S donné), il n'y en a qu'un nombre fini. Ainsi, il sera plus aisé de manipuler les matrices témoins complétées au sens suivant :

[1.K] Définition ***(Matrice*** $(S, f, \mathcal{F})$***-canonique)***

Soient un ensemble $S \subseteq \{0,1\}^k$ *et* M_S *la matrice dont les lignes sont les vecteur* $\vec{a} \in S$.
Soient k'_S *le nombre de collones distinctes de* M_S*, et* k''_S *le nombre de colonnes distinctes de* M_S*, à complémentation près (ie. deux colonnes qui sont l'opposée l'une de l'autre en chaque case sont considérées égales).*
On se donne également une famille de contraintes $\mathcal{F}$ *et une contrainte quelconque* $f^{(k)}$.

- *Si* $\mathcal{F}$ *est* ***héréditaire*** *et* ***non complémentation-close*** *:*
  une matrice W *de taille* $|S| \times (2^{|S|} + k - k'_S)$ *est une* ***matrice*** $(S, f, \mathcal{F})$***-canonique*** *si, quelque soient les* k *premières colonnes, les colonnes restantes sont toutes distinctes* 2 *à* 2 *et distinctes des* k *premières.*
- *Si* $\mathcal{F}$ *est* ***complémentation-close*** *:*
  une matrice W *de taille* $|S| \times (2^{|S|-1} + k - k''_S)$ *est une* ***matrice*** $(S, f, \mathcal{F})$***-canonique*** *si, quelque soient les* k *premières colonnes, les colonnes restantes sont toutes distinctes,* à complémentation près*,* 2 *à* 2 *et des* k *premières.*

Comme prévu, cette notion permet de "capturer" le comportement des gadgets. En effet, vu qu'une matrice canonique présente toutes les colonnes possibles, elle permet d'exprimer toute réduction vers une famille (héréditaire pour pouvoir supprimer les doublons dans la matrice témoin de départ) sous la forme d'un gadget qui lui est associé. C'est ce que formalise le :

[1.L] Lemme

Soient $\alpha \geq 1$*,* $S \subseteq \{0,1\}^k$*, une contrainte* $f^{(k)}$*, et une famille de contraintes* $\mathcal{F}$ *héréditaire.*
On suppose qu'on a un α*-gadget* Γ *qui réduit* f *à* $\mathcal{F}$*, avec la matrice témoin* W.
Alors, pour toute matrice $(S, f, \mathcal{F})$*-canonique* W'*, il existe* $\alpha' \leq \alpha$ *et* Γ' *un* (α', S)*-gadget qui réduit* f *à* $\mathcal{F}$*, et qui admet* W' *pour matrice témoin.*

Preuve de [1.L].

• On se place tout d'abord dans le cas où $\mathcal{F}$ n'est pas complémentation-close (première partie de la définition [1.K]).

Soit $\Gamma = (\vec{Y}, \vec{C}, \vec{w})$ un (α, S)-gadget qui réduit f à $\mathcal{F}$ et soit W une matrice témoin pour Γ. On crée un gadget Γ' avec $n' = 2^{|S|} - k'$ variables secondaires $(Y'_1, \ldots, Y'_{n'})$, chacune associée à une colonne de W' (pas une des k premières).

Pour chaque variable Y_i de Γ, on associe une variable $Z_i \in \{X_1, \ldots X_k, Y'_1, \ldots, Y'_n\}$ telle que la colonne correspondant à Y_i dans W soit identique à celle de Z_i dans W'. Z_i existe bien puisque W' présente toutes les colonnes possibles (c'est une matrice canonique).

À présent, dans chaque clause, on remplace les occurences de Y_i par Z_i. Au cours de ce remplacement, on peut éventuellement introduire deux fois la même variable dans le cas où Z_i et Z_j sont identiques avec $i \neq j$. Il y a donc des clauses où on fait apparaître deux fois la même variable dans les arguments. Mais par la propriété d'hérédité de $\mathcal{F}$, on est certain que la nouvelle clause créée reste dans $\mathcal{F}$.

Au cours du parcours, on peut également arriver à une clause qui est constante égale à $\mathbf{0}$ ou à $\mathbf{1}$. Si $\mathcal{F}$ contient ces deux fonctions, c'est bon. Sinon, il faut justifier que le gadget obtenu n'utilise pas ces fonctions, pour rester dans $\mathcal{F}$. Si on obtient $\mathbf{0}$, on peut simplement supprimer la clause correspondante du gadget puisqu'elle ne sera jamais satisfaite, et donc ne contribuera jamais au poids total. Si on obtient $\mathbf{1}$, on peut également supprimer la clause car pour toute assignation, le poids total sera réduit de w_j qui est une valeur positive. Donc on peut supprimer toutes les clauses $\mathbf{1}$ et on obtient un gadget de performance $\alpha' \leq \alpha$.

Notre construction ne modifie en rien la structure logique du gadget de départ, puisqu'on ne fait que remplacer des variables par d'autres variables ayant les mêmes valeurs de vérité. En particulier, Γ' est effectivement un gadget car toutes les inégalités de la définition [1.F] sont respectées. De plus, W' en est une matrice témoin par construction.

• Voyons maintenant les modifications à apporter à la construction dans le cas d'une famille $\mathcal{F}$ complémentation-close (la définition de matrice $(S, f, \mathcal{F})$-canonique n'étant plus la même).

Ici, chaque colonne Y_i de W se retrouve dans W', mais *à complémentation près*. Si la colonne de Y_i est dans W' sous le nom Z_i, alors on remplace comme ci-dessus Y_i par Z_i. Sinon, c'est que la négation de la colonne de Y_i est dans W' par [1.K] sous le nom Z_i, et alors on remplace Y_i par $\neg Z_i$ dans toutes les clauses. Comme $\mathcal{F}$ est complémentation-close, les fonctions ainsi formées restent dans $\mathcal{F}$. Le reste de la preuve de correction se déroule comme dans le cas héréditaire. □

On peut alors énoncer un corollaire qui nous permet de ramener la recherche de gadget (pour une famille héréditaire) à une recherche parmi un nombre borné universellement de variables auxiliaires. C'est un résultat très important puisqu'a priori, il n'y a pas de méthode naturelle pour trouver un gadget. C'est souvent astucieux.

L'idée sous-jacente est simplement que l'ensemble S étant fini, se donner de plus en plus de variables finit par ne servir à rien puisqu'une nouvelle variable coïnciderait avec une dont on dispose déjà.

[1.M] Corollaire *(Domaine fini de gadgets)*

Soit $f^{(k)}$ une contrainte avec s assignations qui la satisfont, et $\mathcal{F}$ une famille de contraintes. On suppose qu'on dispose d'un α-gadget qui réduit f à $\mathcal{F}$ pour un certain $\alpha \geq 1$. Alors :

- *Si $\mathcal{F}$ est héréditaire, alors il existe un α'-gadget avec au plus $2^s - k'$ variables auxiliaires qui réduit f à $\mathcal{F}$, avec $\alpha' \leq \alpha$, et où k' est le nombre de variables distrinctes parmi les assignations satisfaisant f.*
- *Si $\mathcal{F}$ est complémentation-close, alors il existe un α'-gadget avec au plus $2^{s-1} - k''$ variables auxiliaires qui réduit f à $\mathcal{F}$, avec $\alpha' \leq \alpha$, et où k'' est le nombre de variables distinctes (à complémentation près) parmis les assignations satisfaisant f.*

On a le même résultat en remplaçant s par 2^k et "α'-gadget" par "α'-gadget *strict*" (grâce à [1.G]).

1.3 Recherche automatique de gadgets

Dans la partie précédente, on a vu que pour les familles de contraintes héréditaires, on pouvait restreindre la recherche de gadget à un domaine fini. On voit maintenant une technique pour construire directement un gadget, parfois optimal.

Pour cela, on remarque que les équations dans la définition d'un gadget sont de la forme d'un programme linéaire. Ainsi, on définit le programme linéaire associé à une matrice témoin de la manière suivante :

[1.N] Définition ***(Programme linéaire de recherche de gadget)***

Soit une contrainte $f^{(k)}$, une famille de contraintes $\mathcal{F}$, et une matrice de témoin M de taille $s \times (k+n)$.

On définit le programme linéaire $LP(f, \mathcal{F}, M)$ de la manière suivante :

- *On commence par poser les contraintes $(C_1, \ldots, C_m)$: ce sont toutes les contraintes $f^{(n+k)}$ possibles (donc $m = 2^{2^{k+n}}$).*
- *Les **variables** du programme linéaire sont les poids $(w_1, \ldots, w_m)$, ainsi qu'une variable supplémentaire α.*
- *Soit $S \subseteq \{0,1\}^k$ et $b : S \longrightarrow \{0,1\}^n$ tels que M soit la matrice témoin correspondant à b pour l'ensemble S.*
- *Les **inégalités** (et **égalités**) sont celles de la définition d'un (α, S)-gadget :*

$$(\forall \vec{a} \in \{0,1\}^k,\ f(\vec{a}) = 1)\ (\forall \vec{b'} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b'}) \quad \leq \quad \alpha \tag{11}$$

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 1), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, b(\vec{a})) \quad = \quad \alpha \tag{12}$$

$$(\forall \vec{a} \in \{0,1\}^k,\ f(\vec{a}) = 0)\ (\forall \vec{b'} \in \{0,1\}^n), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, \vec{b'}) \quad \leq \quad \alpha - 1 \tag{13}$$

$$(\forall \vec{a} \in S,\ f(\vec{a}) = 0), \quad \sum_{j=1}^{m} w_j\, C_j(\vec{a}, b(\vec{a})) \quad = \quad \alpha - 1 \tag{14}$$

*On ajoute également les **inégalités** $w_j \geq 0$, pour $j \in [\![1, m]\!]$.*

- *L'**objectif** est de minimiser α.*

La proposition suivante garantit la correction de cette méthode de recherche de gadgets :

[1.O] Proposition ***(Recherche de gadgets par LP)***

Soit une contrainte $f^{(k)}$, une famille de contraintes $\mathcal{F}$, et une matrice témoin M associée à un ensemble $S \subseteq \{0,1\}^k$.

S'il existe un (α, S)-gadget réduisant f à $\mathcal{F}$ avec pour matrice témoin M, alors $LP(f, \mathcal{F}, M)$ trouve un (α', S)-gadget réduisant également f à $\mathcal{F}$, avec α' minimal.

Preuve de [1.O].

Le gadget généré par le programme linéaire possède k variables principales $(X_1, \ldots, X_k)$ puisque k est déterminé par f. Ces variables correspondent aux k premières colonnes de M comme prévu. Puis on a n variables secondaires $(Y_1, \ldots, Y_n)$ qui correspondent aux n colonnes restantes de M. Les contraintes sont toutes celles possibles, et les poids sont renvoyés par le programme linéaire. Par définition, c'est donc un α-gadget réduisant f à $\mathcal{F}$, et α est minimal puisque c'est l'objectif de $LP(f, \mathcal{F}, M)$. □

Nous pouvons maintenant établir le théorème principal à propos de l'existence de gadgets :

[1.P] Théorème ***(Existence et borne sur les gadgets)***

Soit $f^{(k)}$ une contrainte avec s assignations qui la satisfont. Soient k' le nombre de variables distinctes parmi les assignations satisfaisant f, et k'' le nombre de distinctes à complémentation près. Soit $\mathcal{F}$ une famille de contraintes héréditaire, avec des fonctions d'arité au plus l. Alors :

- *S'il existe un α-gadget réduisant f à $\mathcal{F}$, alors il existe un tel gadget avec au plus v variables auxiliaires, où $v = 2^s - k'$ si $\mathcal{F}$ est héréditaire, et $v = 2^{s-1} - k''$ si $\mathcal{F}$ est de plus complémentation-close.*
- *S'il existe un (α, S)-gadget strict réduisant f à $\mathcal{F}$, alors il existe un tel gadget avec au plus v variables auxiliaires, où $v = 2^{2^k} - k'$ si $\mathcal{F}$ est héréditaire, et $v = 2^{2^k - 1} - k''$ si $\mathcal{F}$ est de plus complémentation-close.*

De plus, un tel gadget optimal peut être trouvé en résolvant un programme linéaire avec au plus $|\mathcal{F}| \times (v + k)^l$ variables et 2^{v+k} (in)égalités.

Preuve de [1.P].

D'après [1.M], [1.L] et [1.O], $LP(f, \mathcal{F}, W_S)$ donne un (α, S)-gadget optimal si S est une partie de $\{0,1\}^k$ et W_S est une matrice $(S, f, \mathcal{F})$-canonique.

D'après la remarque [1.G], les choix $S = S_1 = \{\vec{a} \mid f(\vec{a}) = 1\}$ et $S = S_2 = \{0,1\}^k$ mènent respectivement à un α-gadget et à un α-gadget strict, avec α minimal toujours.

Le corollaire [1.M] donne les bornes sur le nombre de variables secondaires. Et la taille du programme linéaire découle directement de sa définition. □

L'objectif est donc atteint : on dispose d'un moyen algorithmique de déterminer (lorsqu'ils existent) des gadgets de réductions, et on a également un moyen mécanique d'obtenir des gadgets certifiés optimaux.

Occupons-nous à présent à utiliser ces résultats sur des familles particulières.

2 Approximabilité de MAXCUT, lien avec PCP

2.1 Équivalence de MAXCUT et MAXCUT/0

On peut voir facilement qu'il est vain de chercher un gadget réduisant un membre de **PC** à **CUT**. Mais nous allons montrer que le problème d'optimisation MAXCUT est équivalent à MAXCUT/0. Il nous suffira donc de chercher des gadgets pour réduire PC_0 et PC_1 à **CUT/0** pour atteindre MAXCUT d'après [1.E].

[2.A] Théorème ***(Équivalence* MAXCUT *et* MAXCUT/0*)***

MAXCUT et MAXCUT/0 sont deux problèmes équivalents.
Plus précisément, si $\mathcal{I}$ est une instance de l'un des deux problèmes, on peut créer une instance $\mathcal{I}'$ de l'autre avec le même optimum et telle qu'une assignation de poids W pour $\mathcal{I}$ peut être transformée en une assignation de même poids pour $\mathcal{I}'$.

Preuve de [2.A].

Tout d'abord, la réduction de MAXCUT à MAXCUT/0 est triviale, puisque la famille de contraintes **CUT/0** contient la famille **CUT**. Donc une instance de **CUT** se transforme en une de **CUT/0** sans changement, et les propriétés du théorème sont évidentes.

Réciproquement, soit une instance $\mathcal{I} = (\vec{X}, \vec{C}, \vec{w})$ de MAXCUT/0 avec n variables et m clauses. On crée une instance $\mathcal{I}' = (\vec{X'}, \vec{C'}, \vec{w})$ de MAXCUT avec $n+1$ variables et toujours m clauses.

Les variables sont simplement les variables $\vec{X}$, avec en plus une variable Z que l'on destine à avoir sa valeur fixée à 0.

Les clauses sont faites comme suit. Les clauses de type CUT sur des variables X_i et X_j sont laissées telles quelles. Si la clause est de la forme $T_0(X_i)$, on la remplace par la clause $CUT(X_i, 0)$.

Voyons maintenant pourquoi cette construction fonctionne. Remarquons d'abord que pour toute assignation $\vec{a'}$ pour le vecteur $\vec{X'}$, $\neg\,\vec{a}$ satisfait les mêmes clauses dans $\mathcal{I}'$. Parmi ces deux assignations équivalentes (en terme de clauses satisfaites), on choisit celle qui fixe la valeur de Z à 0. Il ne reste alors qu'à observer que l'assignation induite par $\vec{a'}$ pour $\vec{X}$ vérifie les clauses de $\mathcal{I}$ correspondant à celles de $\mathcal{I}'$. Comme on a gardé les mêmes poids $\vec{w}$ dans la nouvelle instance, les propriétés de la conclusion du théorème en découlent. □

2.2 Résultat d'inapproximabilité de MAXCUT

CUT/0 est une famille de contraintes héréditaire. En effet, si on identifie les deux variables dans une contrainte CUT, alors on obtient la fonction constante **0**.

Ainsi, par le théorème [1.P], on sait qu'il existe un α-gadget réduisant PC_0 à **CUT/0** avec au plus 13 variables auxiliaires (il y a 3 variables primaires distinctes, et $s = 4$ assignations qui valident PC_0). En particulier, on peut utiliser la résolution par programmation linéaire pour trouver un gadget optimal. En appliquant cette technique on obtient le :

[2.B] Lemme

*Il existe un 8-gadget réduisant PC_0 à **CUT/0**. Il est optimal et strict.*
*Il existe un 9-gadget réduisant PC_1 à **CUT/0**. Il est optimal et strict.*

Et en appliquant le théorème [1.E], on obtient ler :

[2.C] Théorème ***(Inapproximabilité de* MAXCUT*)***

Pour tout $\varepsilon > 0$, le problème d'optimisation MAXCUT *est difficile à approximer dans un facteur $16/17 + \varepsilon$.*

Preuve de [2.C].
Le lemme [2.B] associé au théorème [1.E] donne que **MAXCUT/0** est difficile à approximer dans un facteur $16/17 + \varepsilon$ pour tout $\varepsilon > 0$.
Alors le théorème [2.A] permet de conclure. □

2.3 Lien avec PCP

Ce résultat a des implications pour la classe **PCP**. Voir [TSSW00, p. 3] qui renvoie à [BGS98] pour plus de détails.

3 Approximabilité de MAX-2CSP, lien avec PCP

3.1 Inapproximabilité de MAX-2CSP

La technique de la programmation linéaire permet de trouver que :

[3.A] Lemme
Il existe des 5*-gadgets qui réduisent* PC_0 *et* PC_1 *à* ***2CSP****, et ils sont optimaux et stricts.*

D'où le :

[3.B] Théorème ***(Inapproximabilité de*** **MAX-2CSP*****)***
Pour tout $\varepsilon > 0$*, le problème d'optimisation* MAX-2CSP *est difficile à approximer dans un facteur* $9/10 + \varepsilon$*.*

3.2 Lien avec PCP

Le théorème [3.B] permet de déduire le :

[3.C] Théorème
Pour tout $\varepsilon > 0$*, il existe des constantes* c *et* s *telles que* $\mathbf{NP} \subseteq \mathbf{PCP}_{\mathbf{c},\mathbf{s}}[\log, 2]$ *et* $c/s > 10/9 - \varepsilon$*.*

On renvoie à [TSSW00, p.16] pour plus de détails.

4 Approximabilité de MAX-3ConjSAT, lien avec PCP

4.1 Inapproximabilité de MAX-3ConjSAT

En construisant des gadgets toujours avec la même technique des contraintes de MAX-3ConjSAT vers **2SAT**, on peut montrer le résultat suivant (qui demande plus de travail : il faut passer de l'existence de gadget à un algorithme d'approximation, et pas à la non-approximation comme jusqu'ici) :

[4.A] Théorème ***(Approximabilité de* MAX-3ConjSAT*)***
Il existe un algorithme polynômial pour approximer ***MAX-3ConjSAT*** *dans un rapport de* 0.367*.*

Idem, on renvoie à [TSSW00, p. 18] pour plus de détails.

4.2 Lien avec PCP

Le théorème [4.A] a pour conséquence notoire le :

[4.B] Théorème
$\mathbf{PCP}_{\mathbf{c},\mathbf{s}}[\log, 3] \subseteq \mathbf{P}$ *pour* $c/s > 2.7214$.

Dans [TSSW00, p. 21], on renvoie à [Tre98, théorème 18] pour le passage de [4.A] à [4.B].

Références

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and nonapproximability - toward tight results. *SIAM J. on computing*, 27(3) :pp. 804–915, 1998.

[CST96] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight : Where is the question ? *Proc. of the 4th Israel Symposium on Theory of Computing and Systems*, pages pp. 68–77, 1996.

[Has97] Johan Hastad. Some optimal inapproximability results. *Proc. of the 29th ACM Symposium on Theory of computing*, 1997.

[Tre98] Luca Trevisan. Parallel approximation algorithms using positive linear programming. *Algorithmica*, 21 :pp. 7, 1998.

[TSSW00] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approxmation, and linear programming. *SIAM J. on computing*, 29(6) :pp. 2074–2097, 2000.