Complexité paramétrée et kernelization pour les problèmes de satisfaction de contraintes

Mémoire de fin d'étude

Master *Sciences et Technologies*,
Mention *Informatique*,
Parcours MOCA

Auteur
Benjamin Bergougnoux

Superviseurs
Christophe Paul
Philippe Janssen

Lieu de stage
LIRMM UM5506 - CNRS, Université de Montpellier

version 1 soutenu publiquement le 12 Juin 2015

Résumé

Ce stage de master est une analyse de problèmes combinatoires au cœur de l'intelligence artificielle avec les nombreux outils proposés par la complexité paramétrée. Nous nous focaliserons sur le problème de satisfaction de contraintes (CSP) et sur les contraintes globales.

Les trois types de résultats sur lesquels nous travaillerons sont ceux concernant la classe de complexité FPT, les kernelizations et l'hypothèse ETH (exponential-time hypothesis).

Ce mémoire de fin d'étude présente les résultats obtenus au cours de ce stage.

Parmi ces résultats figurent l'inexistence de noyau polynomial pour les CSP booléens paramétrés par le nombre de variables et le nombre de contraintes et l'existence d'un algorithme FPT pour la consistance de la contrainte globale Common.

Abstract

This master thesis is a analyse of some famous combinatorial problems in AI with the tools provided by the parameterized complexity. We consider problems from constraint satisfaction (CSP) and global constraints.

We work on three kinds of results : those about the complexity class FPT, those who concern kernelization and those based on the exponential-time hypothesis (ETH).

This master thesis presents the results obtained during this internship.

Among those results there are the inexistence of a polynomial kernel for the boolean CSP parameterized by the number of variables and the number of constraints and there are also a FPT algorithm for the consistency of the global constraint Common.

Table des matières

1

Introduction

Les problèmes $\mathcal{NP}$-complets sont omniprésents dans la recherche en informatique théorique, ils ont de nombreuses applications et permettent de modéliser de nombreux problèmes pratiques.

À ce jour, aucun algorithme efficace (i.e. dont le temps d'exécution est polynomial en la taille de la donnée) n'a été trouvé permettant de résoudre un problème $\mathcal{NP}$-complet. L'existence d'un tel algorithme impliquerait la résolution en temps polynomial de l'ensemble des problèmes de la classe $\mathcal{NP}$.

Des décennies de recherche infructueuses ont motivé la formulation de la conjecture $\mathcal{P} \neq \mathcal{NP}$. Cette conjecture stipule qu'il n'existe pas d'algorithme polynomial permettant de résoudre les problèmes $\mathcal{NP}$-complets. Résoudre cette conjecture fait partie des problèmes mathématiques les plus difficiles et importants du siècle [13].

Introduite par Downey et Fellows [15] à la fin des années 80, la complexité paramétrée est une branche émergente de la théorie de la complexité permettant une analyse très fine des problèmes $\mathcal{NP}$-difficiles en prenant en compte des paramètres additionnels à la taille de la donnée.

Devant un problème $\mathcal{NP}$-difficile dont on souhaite une solution exacte, la complexité paramétrée offre de nombreux outils permettant de trouver des algorithmes exacts et « rapides ». L'objectif est de trouver un paramètre capturant toute la difficulté du problème et de trouver un algorithme dont le temps d'exécution est exponentiel uniquement en fonction de ce paramètre.

Ces algorithmes dit *fixed-tractable parameter* permettent de résoudre des problèmes $\mathcal{NP}$-difficiles sur des instances arbitrairement grandes tant que la valeur du paramètre est raisonnable.

La complexité paramétrée a commencé à se développer essentiellement dans les problèmes liés aux graphes. Ce n'est que récemment qu'elle fut appliquée aux problèmes liés à l'IA [3, 20, 21, 22].

Les problèmes de satisfaction de contraintes occupent une place importante dans l'IA mais aussi dans la recherche opérationnelle [9] : ils permettent en effet de modéliser de nombreux problèmes réels comme les problèmes d'ordonnancement, ceux d'emplois du temps ou encore le problème de *car sequencing*.

Ce stage de Master planifié sur trois mois se déroule au LIRMM sous l'encadrement de Christophe Paul et Philippe Janssen.

Il fut précédé par une étude bibliographique de sept semaines, qui avait pour objectif de faire un état de l'art et de trouver des pistes de recherche intéressantes pour le stage.

Les résultats du stage suivent ces pistes et sont bien dans la continuité des résultats abordés dans l'étude bibliographique.

Notons qu'une des questions ouvertes que j'aborde dans mon étude bibliographique provenant de l'article [22], sur la complexité paramétrée de la recherche d'une backdoor pour la classe de formule SAT nommée « renamable horn », n'était en réalité plus ouverte, elle fut résolue par Gaspers et Szeider dans l'article [19].

Les résultats que nous avons trouvés concernent le problème CSP classique, où les contraintes sont données en extension et sur la consistance de certaines contraintes globales. Voici une synthèse des résultats trouvés au cours de ce stage :

- Sauf si $\mathcal{NP} \subseteq \text{co}\mathcal{NP}/\text{poly}$ [1], il n'existe pas de noyau polynomial pour les problèmes suivants :
  - CSP_{bool} paramétré par la largeur de chemin du graphe primal et du graphe dual et par la profondeur. Ce résultat est un renforcement d'un résultat de Gaspers et Szeider l'article [20].
  - 2-CSP paramétré par la taille d'un vertex-cover minimum du graphe primal, la taille du domaine et la profondeur.
  - 2-CSP paramétré par la taille d'un linear-feedback-vertex set minimum du graphe primal même quand le domaine est fixé à 3 et la profondeur à 6.
  - CSP_{bool} paramétré par le nombre de variables et de contraintes.
  - Cons-Among, Cons-Roots et Cons-Common[2] paramétrés par la taille de l'union des domaines.
- Le problème Cons-Common est FPT paramétré par $|\mathcal{D}(X) \cap \mathcal{D}(Y)|$ [3].
- Sauf si ETH [4] est fausse, les problèmes Cons-Among, Cons-Roots et Cons-Common ne peuvent être résolu en temps $O^*(2^{o(\boldsymbol{dom})})$ où $\boldsymbol{dom}$ est la taille de l'union des domaines.

Ce stage fut une expérience très enrichissante. J'ai pu acquérir et approfondir des notions en complexité paramétrée et en théorie des graphes.

J'ai eu la chance de découvrir l'ambiance torride d'un laboratoire de recherche, d'assister à de nombreux séminaires et de me faire une meilleure idée du métier de chercheur et du travail que représente une thèse.

[1] Hypothèse de complexité.

[2] Notation du problème de consistance sur une contrainte globale, les contraintes globales Among, Roots et Common sont définies dans le chapitre 4.

[3] Ce paramètre est défini dans le chapitre 4.

[4] Une autre hypothèse de complexité.

1.1 Plan à venir

Dans le prochain chapitre, nous verrons de nombreuses notions nécessaires à la compréhension des résultats que nous aborderons dans ce mémoire. Nous commencerons par les notions propres à la complexité paramétrée et à la kernelization puis nous présenterons l'hypothèse de complexité ETH.

Dans le chapitre 3, on définit le problème CSP et nous présentons les bornes inférieures concernants la kernelization de différentes paramétrisations de ce problème.

Dans le chapitre 4, nous présenterons la notion de contraintes globales et nos résultats qui les concernent. Nous verrons également un problème que nous n'avons pas pu résoudre mais dont les implications sont très intéressantes.

2

Préliminaires

2.1 Classes de problèmes paramétrés

La complexité paramétrée étudie les problèmes *paramétrés*, formellement un problème paramétré est un langage $L \subseteq \Sigma^* \times \mathbb{N}$, où Σ est un alphabet fini. Dans une instance d'un problème paramétré $(x, k) \in \Sigma^* \times \mathbb{N}$, la seconde composante k est appelé le paramètre.

La classe de complexité XP est constituée de tous les problèmes paramétrés qui sont polynomiaux quand leur paramètre est fixé. Plus formellement un problème paramétré L est XP ssi $(x, k) \in L$ est décidable en temps $f(k) \cdot (|x| + k)^{g(k)}$ où g et f sont des fonctions calculables.

Par exemple, k-Clique et k-Vertex Cover [1] sont dans XP, en effet ces deux problèmes sont décidables en temps $O(n^k)$, ce qui n'est pas le cas de k-SAT [2] qui est $\mathcal{NP}$-complet pour $k \geq 3$.

Un des objectifs central de la complexité paramétrée est de trouver des algorithmes dit *fixed-parameter tractable*, c'est à dire dont le temps d'exécution est exponentiel uniquement en fonction du paramètre.

Par exemple, le problème k-Vertex Cover est solvable en temps $O(1,271^k + kn)$ [12]. Les problèmes admettant de tel algorithme sont dit FPT.

Plus formellement, un problème paramétré L est FPT ssi on peut décider si $(x, k) \in L$ en temps $f(k) \cdot (|x| + k)^{O(1)}$ où f est une fonction calculable. Il existe de nombreux problèmes XP qui ne sont vraisemblablement pas FPT, c'est le cas notamment de k-Clique.

Pour plus d'explications et de détails sur les classes de complexité XP et FPT, le lecteur peut s'orienter sur l'article [16].

Similairement à la théorie de la $\mathcal{NP}$-complétude, il existe dans la complexité paramétrée une hiérarchie appelée W-hiérarchie. Cette hiérarchie se compose des classes de complexité $\mathrm{W}[0], \mathrm{W}[1], \ldots, \mathrm{W}[P]$ et forme la chaine :

$$\mathrm{FPT} = \mathrm{W}[0] \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots \subseteq \mathrm{W}[P] \subseteq \mathrm{XP}$$

[1] Paramétrés par la taille de la solution.
[2] Paramétré par la taille maximale des clauses.

Analogiquement à la conjecture $\mathcal{P} \neq \mathcal{NP}$, on conjecture que FPT $\neq$ W[1], ces deux conjectures ont de nombreuses similarités et la deuxième est presque aussi raisonnable que la première [16].

Il est a noté que le problème paramétré par k consistant à savoir si une machine de Turing indéterministe s'arrête en k étapes est W[1]-complet.

Naïvement, on peut résoudre ce problème en temps $O(n^k)$ [3] en explorant exhaustivement l'arbre de degré n et de profondeur k des chemins de calculs possibles.

Il semble improbable qu'on puisse résoudre ce problème sans faire cette exploration exhaustive et il est convenable donc de penser que FPT $\neq$ W[1].

Jouant le même rôle que la réduction polynomiale dans la $\mathcal{NP}$-complétude, la réduction paramétrée est la relation d'ordre au sein de la W-hiérarchie [17]. Elle permet également de définir la relation d'équivalence permettant de définir les classes W[1], W[2], ..., W[P]. Sa définition est la suivante :

Définition 1. *Soit L, L' deux problèmes paramétrés, il existe une réduction paramétrée de L vers L' ssi il existe deux fonctions calculables f et g et un algorithme dont le temps d'exécution est $f(k) \cdot n^{O(1)}$. Cet algorithme associe à chaque instance (x, k) de L une instance (x', k') de L' telle que :*

- *$(x, k) \in L$ si et seulement si $(x', k') \in L'$,*
- *$k' \leq g(k)$.*

L'existence d'une telle transformation de L vers L' est notée : $L \leq_{\text{FPT}} L'$.

Théorème 1 ([17])**.** *Soit L, L' deux problèmes paramétrés tels que $L \leq_{\text{FPT}} L'$.*
Si L' est FPT alors L l'est aussi.

Pour montrer qu'un problème paramétré L n'est pas FPT, il faut exhiber une réduction paramétrée d'un problème W[1]-difficile vers L. Ceci démontrera que L n'est pas FPT sauf si W[1] = FPT.

Les problèmes k-Clique et k-Multicolored-Clique [4] sont deux problèmes W[1]-complets souvent utilisés dans les réductions paramétrées pour prouver la W[1]-difficulté d'un problème paramétré [3, 27].

Dans nos résultats les problèmes sont parfois paramétrés par un ensemble de paramètres. Les définitions abordées plus haut peuvent aisément se généraliser à cette notion.

Par exemple, un problème paramétré L par $\{k_1, k_2, \ldots, k_t\}$ est FPT ssi il existe un algorithme décidant $(x, k_1, \ldots, k_t) \in L$ en temps $f(k_1, \ldots, k_t) \cdot (|x| + k_1 + \cdots + k_t)^{O(1)}$ où f est une fonction calculable.

La version non-paramétrée d'un problème paramétré $L \subseteq \Sigma^* \times \mathbb{N}$ est un problème $P = \{x\#1^k : (x, k) \in L\}$, où $\#1^k$ est une écriture en unaire du paramètre k. Cette notion est utilisée dans les deux derniers théorèmes de la section suivante.

[3] Ici n est la taille de la machine de Turing.
[4] Paramétrés par la taille de la solution.

2.2 Kernelization

La kernelization est une branche de la complexité paramétrée, elle permet d'étudier les méthodes de preprocessing dans un cadre rigoureux et avec des garanties de performance, formellement, on dit qu'un problème paramétré L admet une kernelization si il admet un algorithme qui étant donné $(x, k) \in \Sigma^* \times \mathbb{N}$ renvoie en temps polynomial en $|x| + k$, une instance $(x', k') \in \Sigma^* \times \mathbb{N}$ telle que :

- $(x, k) \in L$ si et seulement si $(x', k') \in L$,
- $|x'|, k' \leq g(k)$ où g est une fonction calculable.

L'instance renvoyée par l'algorithme est appelé un noyau, g est la taille du noyau et si $g(k) \leq k^{O(1)}$ alors on dit que L admet un noyau polynomial.

Tous les problèmes FPT admettent une kernelization, de fait ces deux notions sont équivalentes [16].La kernelization est une autre alternative pour définir la classe FPT, cependant être FPT n'implique pas d'avoir un noyau polynomial.

Par exemple, les problèmes Longest-Path [5] et SAT[6] sont deux problèmes FPT n'admettant pas noyau polynomial.

La transformation paramétrée polynomiale est une variante la transformation paramétrée, elle permet de transférer la propriété « possède un noyau polynomial » d'un problème à un autre. Voici sa définition :

Définition 2. *Soient $L, L' \subseteq \Sigma^* \times \mathbb{N}$ deux problèmes paramétrés. Il existe une transformation paramétrée polynomiale de L vers L' ssi il existe un polynôme p et un algorithme polynomial [7] qui associe à une instance $(x, k) \in \Sigma^* \times \mathbb{N}$ une instance $(x', k') \in \Sigma^* \times \mathbb{N}$ telle que :*

- *$(x, k) \in L$ ssi $(x', k') \in L'$,*
- *$k' \leq p(k)$.*

L'existence d'une telle transformation de L à L' est notée : $L \leq_{\text{TPP}} L'$.

Théorème 2 ([6])**.** *Soit L, L' deux problèmes paramétrés tels que $L \leq_{\text{TPP}} L'$.*
Si L' admet un noyau polynomial alors L en admet un aussi.

Dans ce stage, nous avons uniquement des résultats négatifs sur les noyaux polynomiaux, la transformation paramétrée polynomiale permet de trouver de tels résultats, il suffit de prendre pour problème de départ un problème connu pour ne pas avoir de noyau polynomial.

Les résultats négatifs concernant les noyaux polynomiaux se basent tous sur l'hypothèse $\mathcal{NP} \not\subseteq \text{co}\mathcal{NP}/\text{poly}$, cette hypothèse n'est pas aussi forte que $\mathcal{P} \neq \mathcal{NP}$ mais elle est jugée solide par la communauté scientifique [31].

Si cette hypothèse s'avérait fausse, cela impliquerait en particulier un effondrement de la hiérarchie polynomiale à son troisième niveau : $\text{PH} = \Sigma_p^3$.

[5] Paramétré par la taille de la solution.
[6] Paramétré par le nombre de variables.
[7] Polynomial en $(|x| + k)$.

Il existe d'autres outils pour prouver qu'un problème paramétré n'admet pas de noyau polynomial, dans ce stage nous utiliserons deux d'entre eux, la OU-composition et la composition croisée. Ci-dessous figurent leurs définitions formelles et les théorèmes qui leurs sont associés.

La OU-composition fut introduite par Bodlaender et al. [6]. C'est le premier outil développé permettant de prouver l'inexistence d'un noyau polynomial. Le théorème [8] permettant d'utiliser cet outil se base sur un résultat de Fortnow et Santhanam [18].

L'idée de la OU-composition est de compresser une séquence d'instances d'un problème paramétré en une nouvelle instance de ce problème. Les instances de cette séquence ont des paramètres égaux et le paramètre de l'instance d'arrivée est polynomialement borné par le paramètre des instances de départ.

Définition 3 (OU-composition)**.** *Un algorithme de OU-composition pour un problème paramétré $L \subseteq \Sigma^* \times \mathbb{N}$ est un algorithme qui reçoit une séquence $((x_1, k), \ldots, (x_t, k))$, telle que $(x_i, k) \in \Sigma^* \times \mathbb{N}$ pour tout $i \in [1, t]$, cet algorithme possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$ et retourne $(y, k') \in \Sigma^* \times \mathbb{N}$ tel que :*

- $(y, k') \in L \Leftrightarrow \exists i \in [1, t]$ *tel que* $(x_i, k) \in L$,
- k' *est polynomial en* k.

Théorème 3 ([8])**.** *Un problème paramétré OU-composable dont la version non-paramétrée est $\mathcal{NP}$-complet n'admet pas de noyau polynomial sauf si $\mathcal{NP} \subseteq \text{co}\mathcal{NP}/\text{poly}$.*

La composition croisée est un outil plus général que la OU-composition pour trouver des bornes inférieures pour la kernelization, elle fut introduite en 2014 dans un article de Bodlaender et al [7].

Cette transformation prends en entrée une séquence d'instances d'un problème non-paramétré et équivalentes deux à deux par rapport à une relation d'équivalence dite polynomiale. Elle renvoie une instance d'un problème paramétré dont le paramètre est borné polynomialement par la taille maximum des instances de départ et par le logarithme du nombre d'instances de départ.

La OU-composition est une composition croisée pour une certaine relation d'équivalence polynomiale.

Ci-dessous figure la définition formelle de la composition croisée et celle de la relation d'équivalence polynomiale ainsi que le théorème associé à la composition croisée.

Définition 4 ([7])**.** *Une relation d'équivalence $\mathcal{R}$ sur Σ^* est une relation d'équivalence polynomiale si il existe une algorithme qui décide si $x\mathcal{R}y$ en temps $(|x| + |y|)^{O(1)}$ et si pour tout sous-ensemble $S \subseteq \Sigma^*$, $\mathcal{R}$ partitionne S en au plus $(\max_{Y \in S} |Y|)^{O(1)}$ classes d'équivalence.*

Soit un problème $L \subseteq \Sigma^*$ et un problème paramétré $L' \subseteq \Sigma^* \times \mathbb{N}$. Il existe une composition croisée de L vers L' si il existe une relation d'équivalence polynomiale $\mathcal{R}$ sur Σ^* et un algorithme $\mathcal{A}$ qui :

- *Reçoit une séquence $(x_1, \ldots, x_t)$, telle que $\forall\ 1 \leq i \leq j \leq t$, $x_i \mathcal{R} x_j$,*
- *Possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,*

- *Retourne* $(x,k) \in \Sigma^* \times \mathbb{N}$ *tel que* $(x,k) \in L' \Leftrightarrow\ \exists i \in [1,t],\ x_i \in L$,
- k *est polynomial en* $\max_{i=1}^{t} |x_i| + \log(t)$.

Théorème 4 ([7]). *Si il existe une composition croisée d'un problème $\mathcal{NP}$-complet $L \subseteq \Sigma^*$ vers un problème paramétré $L' \subseteq \Sigma^* \times \mathbb{N}$, tel que la version non-paramétrée de L' soit $\mathcal{NP}$-complet alors L' n'admet pas de noyau polynomial, sauf si $\mathcal{NP} \subseteq$ co$\mathcal{NP}$/poly.*

2.3 Hiérarchie de paramètres

Dans l'étude bibliographique, nous avions déjà abordé la notion de dominance entre paramètres. Un paramètre k domine un paramètre k' ssi il existe une fonction calculable f telle que pour toute instance du problème étudié : $k' \leq f(k)$.

Si k domine k' et si un problème P paramétré par k' est FPT alors P paramétré par k est FPT.

Chercher le paramètre le plus dominé tel que le problème soit FPT est un objectif central quand on analyse la complexité paramétrée d'un problème.

La hiérarchie de paramètres s'applique également quand on étudie la kernelization mais ici la dominance entre paramètres doit être polynomiale. Un paramètre k domine un paramètre k' ssi il existe un polynôme p tel que pour toute instance du problème étudié : $k' \leq p(k)$.

De même, si k domine k' et si un problème P paramétré par k' admet un noyau polynomial alors P paramétré par k admet également un noyau polynomial.

Nous ne distinguerons pas les deux notions de dominance dans ce rapport, le contexte étant à chaque fois explicite, il est simple de savoir de laquelle des deux nous parlons. D'ailleurs nous parlerons presque exclusivement de la notions de dominance polynomiale.

Cette notion de dominance est généralisable à des ensembles de paramètre. Un ensemble S de paramètres domine un ensemble de paramètres $S' = \{k'_1, \ldots, k'_t\}$ ssi il existe un polynôme p tel que pour tout paramètre $k \in S$ et pour toute instance du problème étudié : $k \leq p(k'_1, \ldots, k'_t)$.

2.4 Exponential-Time Hypothesis

Certains problèmes $\mathcal{NP}$-complets admettent des algorithmes sous-exponentiels (i.e. en temps $O^*(2^{o(n)})$ [8]), c'est le cas par exemple du problème Independent Set dans les graphes planaires qui peut être résolu en temps $O^*(2^{\sqrt{n}})$ [9] [28].

Inversement d'autres problèmes ne semblent vraisemblablement pas en admettre.

En 50 ans de recherche, aucun algorithme sous-exponentiel n'a été trouvé pour résoudre k-SAT même dans le cas le plus simple où $k = 3$.

Il semble donc raisonnable de penser qu'il n'existe pas tel algorithme pour le problème k-SAT. L'hypothèse $\mathcal{P} \neq \mathcal{NP}$ n'impliquant pas ce fait, Impagliazzo et Paturi ont proposé en 2001, une hypothèse plus forte appelée ETH [25] dont l'énoncé formel est le suivant :

[8] O^* est une notation d'une complexité ignorant les termes polynomiaux. Une fonction appartient à $o(n)$ ssi elle est dominée asymptotiquement de façon stricte par une fonction linéaire.

[9] Ici n est le nombre de sommets du graphe.

Hypothèse 1 (ETH)**.** *Soit δ_k l'infinimum de l'ensemble des constantes c telles qu'il existe un algorithme résolvant k-*SAT *en temps $O^*(2^{cn})$. Pour tout $k \geq 3$, $\delta_k > 0$.*

L'hypothèse ETH a permis de montrer l'optimalité de nombreux algorithmes exponentiels classiques, mais aussi d'algorithmes XP et FPT [29].

Par exemple, sauf si ETH est fausse, le problème Vertex Cover ne peut pas être résolu en temps $2^{o(k)}$ et le problème Clique ne peut pas être résolu en temps $n^{o(k)}$ [10] [11].

Remarquons que l'hypothèse de complexité paramétrée FPT $\neq$ W[1] est une implication d'ETH.

Il existe une variante forte de ETH appelé SETH stipulant que $\lim_{k\to\infty} \delta_k = 1$. SETH implique ETH mais elle n'est pas jugée aussi fiable que cette dernière par la communauté scientifique.

Pour trouver des résultats avec ETH, il est nécessaire de définir une réduction préservant la propriété « solvable en temps sous-exponentiel ».

Il existe dans la littérature plusieurs réductions préservant la propriété « solvable en temps sous-exponentiel », l'une des plus générales étant la *serf-Turing* réduction aussi appelée *serf-T* réduction [29].

Nous avions défini cette réduction dans l'étude bibliographique, mais nous ne l'utilisons pas dans nos résultats. Nous utilisons une réduction beaucoup plus simple nommée réduction paramétrée linéaire, dont la définition est la suivante :

Définition 5. *Soient $L, L' \subseteq \Sigma^* \times \mathbb{N}$ deux problèmes paramétrés. Il existe une transformation paramétrée linéaire de L vers L' ssi il existe une constante c et un algorithme polynomial qui associe à une instance $(x, k) \in \Sigma^* \times \mathbb{N}$ une instance $(x', k') \in \Sigma^* \times \mathbb{N}$ telle que :*

- *$(x, k) \in L$ si et seulement si $(x', k') \in L'$,*
- *$k' \leq c \times k$.*

Théorème 5. *Soient $L, L' \subseteq \Sigma^* \times \mathbb{N}$ deux problèmes paramétrés tels qu'il existe une réduction paramétrée linéaire de L vers L'.*

Si L' admet un algorithme sous-exponentiel dont le temps d'exécution est $O^*(2^{o(k')})$ alors L en admet un algorithme sous-exponentiel dont le temps d'exécution est $O^*(2^{o(k)})$

[10] n est le nombre de sommet du graphe et k la taille de la solution.

3

Kernelization du CSP : bornes inférieures

3.1 Définitions du problème

Formellement une instance du problème CSP est un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, où $\mathcal{V}$ est un ensemble fini de variables, $\mathcal{C}$ est un ensemble fini de contraintes et $\mathcal{D}$ est une fonction assignant à chaque variable de $\mathcal{V}$ un ensemble de valeurs, $\mathcal{D}(x)$ est appelé le domaine de la variable x. Soit $X \subseteq \mathcal{V}$, notons $\mathcal{D}(X)$ l'union des domaines des variables de X.

Chaque contrainte C est définie par une séquence de variables distinctes de $\mathcal{V}$ appelée portée notée $\mathit{scope}(C)$ et par un ensemble de tuples autorisés noté $\mathit{list}(C)$. L'ensemble $\mathit{list}(C)$ est composé de tuples de longueur $|\mathit{scope}(C)|$.

Soi une variable $x \in \mathcal{V}$, notons $\boldsymbol{cons}(x) = \{C \in \mathcal{C} \mid x \in \mathit{scope}(C)\}$ c'est l'ensemble des contraintes qui portent sur x.

Une assignation α est une fonction de $\mathcal{V} \to \mathcal{D}(\mathcal{V})$ telle que $\alpha(x) \in \mathcal{D}(x)$ pour tout $x \in \mathcal{V}$. Une assignation α satisfait une contrainte C où $\mathit{scope}(C) = (x_1, \ldots, x_r)$ ssi $(\alpha(x_1), \ldots, \alpha(x_r)) \in \mathit{list}(C)$.

Le problème CSP consiste à savoir si il existe une assignation satisfaisant toutes ses contraintes.

Définissons quelques paramètres concernant le CSP, pour une instance $I = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ donnée :

- Le nombre de variables $\boldsymbol{vars} = |\mathcal{V}|$,
- Le nombre de contraintes $\boldsymbol{cons} = |\mathcal{C}|$,
- La taille des domaines $\boldsymbol{dom} = |\mathcal{D}(\mathcal{V})|$,
- La profondeur maximum $\boldsymbol{dep} = \max_{C \in \mathcal{C}} |\mathit{list}(C)|$,
- L'arité maximum $\boldsymbol{arity} = \max_{C \in \mathcal{C}} |\mathit{scope}(C)|$,
- Le degré maximum d'une variable $\boldsymbol{deg} = \max_{x \in \mathcal{V}} |\mathit{con}(x)|$.

Nous noterons $\mathrm{CSP}_{\mathrm{bool}}$ la restriction du problème CSP aux instances où $\boldsymbol{dom} = 2$ et 2-CSP la restriction du problème CSP aux instances où $\boldsymbol{arity} = 2$.

Ces deux sous-problèmes sont $\mathcal{NP}$-complets. Il existe des réductions polynomiales simples de 3-SAT vers CSP_{bool} et du problème de la coloration de graphe vers 2-CSP. Par contre, 2-CSP_{bool} est un problème polynomial, on peut le réduire à 2-SAT.

La complexité paramétrée est très développée sur les problèmes concernant les graphes, il est donc intéressant d'avoir un lien entre les graphes et le problème CSP pour bénéficier des nombreux paramètres existants dans les graphes. Ainsi à une instance $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ du problème CSP, on peut lui associer l'hypergraphe et les trois graphes suivants :

- L'hypergraphe des contraintes $H = (\mathcal{V}, \{scope(C) : C \in \mathcal{C}\})$.
- Le graphe primal $G = (\mathcal{V}, \{(x, y) : con(x) \cap con(y) \neq \varnothing\})$.
- Le graphe dual $G^d = (\mathcal{C}, \{(C_1, C_2) : scope(C_1) \cap scope(C_2) \neq \varnothing\})$.
- Le graphe biparti d'incidence $G^* = (\mathcal{V} \cup \mathcal{C}, \{(x, C) : x \in scope(C)\})$.

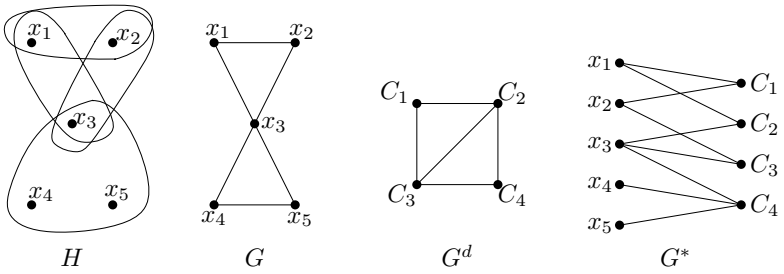

FIGURE 3.1: Cette figure représente l'hypergraphe et les graphes dont on parle pour une instance où $\mathcal{V} = \{x_1, \ldots, x_5\}$ et $\mathcal{C} = \{C_1, \ldots, \mathcal{C}_4\}$ où $scope(C_1) = \{x_1, x_2\}$, $scope(C_2) = \{x_1, x_3\}$, $scope(C_3) = \{x_2, x_3\}$ et $scope(C_4) = \{x_3, x_4, x_5\}$.

3.2 Largeur arborescente et largeur de chemin

La largeur arborescente ou treewidth en anglais est un paramètre très utilisé et permettant à de nombreux problèmes sur les graphes d'être FPT [17].

De fait le théorème de Courcelle [14] stipule que n'importe quelle propriété sur les graphes exprimable en logique monadique du second ordre peut être testée en temps $f(k) \cdot n$ sur les graphes de largeur arborescente au plus k. Ci-dessous figure une définition formelle de la largeur arborescente.

Définition 6. *Soit $G = (V, E)$ un graphe, une décomposition arborescente de G est une paire (T, χ) où T est un arbre et une χ une fonction assignant à chaque nœud de T un sous-ensemble de V tel que les propriétés suivantes soient satisfaites :*

- *Pour tout sommet $v \in V$, il existe un nœud t de T tel que $v \in \chi(t)$,*
- *Pour toute arête $(u, v) \in E$, il existe un nœud t de T tel que $u, v \in \chi(t)$,*

- *Pour tout nœud t_1, t_2, t_3 de T, si t_2 appartient à l'unique chemin entre t_1 et t_3 alors $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$ (i.e. pour tout sommet x de G l'ensemble des nœuds de T contenant x est connexe).*

La largeur d'une décomposition arborescente (T, χ) est le maximum $|\chi(t)| - 1$ pour tous les nœuds t de T.

La largeur arborescente d'un graphe est la largeur minimum de toutes ses décompositions arborescentes. La largeur arborescente d'un graphe G est notée $\boldsymbol{tw}(G)$.

Notons $\mathbf{tw}, \mathbf{tw}^d$ et $\mathbf{tw}^*$ la largeur arborescente respective du graphe primal, du graphe dual et du graphe d'incidence.

Nous avons vu que le problème CSP paramétré par $\boldsymbol{dom}$ et $\mathbf{tw}$ est un problème FPT [33] mais n'admet pas de noyau polynomial même quand $\boldsymbol{dom} = 2$. Ce dernier résultat figure dans l'article [20] de Gaspers et Szeider.

Comme nous allons le voir, il est possible d'obtenir avec leur preuve, un résultat plus fort en prenant en compte la largeur de chemin [1] et non la largeur arborescente.

Avant tout, définissons formellement la largeur de chemin :

Définition 7 (Largeur de chemin, décomposition en chemin)**.** *Soit $G = (V, E)$ un graphe, une décomposition en chemin de G est une décomposition arborescente (P, χ) de G où P est un chemin.*

La largeur d'une décomposition en chemin (P, χ) est le maximum $|\chi(p)| - 1$ pour tous les nœuds p de P.

La largeur de chemin d'un graphe est la largeur minimum de toutes ses décompositions en chemin. La largeur de chemin d'un graphe G est notée $\boldsymbol{pw}(G)$.

Ci-dessous figure un exemple de décomposition arborescente et de décomposition en chemin d'un graphe.

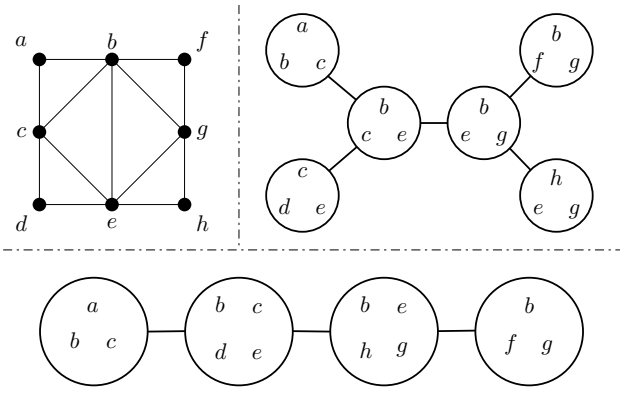

FIGURE 3.2: Décomposition arborescente et en chemin d'un graphe.

Notons respectivement $\mathbf{pw}$ et $\mathbf{pw}^d$ la largeur de chemin du graphe primal et celle du graphe dual. Par définition, la largeur de chemin d'un graphe est toujours plus grande que sa largeur arborescente, donc $\mathbf{pw}$ domine $\mathbf{tw}$ et $\mathbf{pw}^d$ domine $\mathbf{tw}^d$. C'est pourquoi le théorème suivant est plus fort que celui de Gaspers et Szeider.

[1] Plus connu sous le nom Pathwidth.

Théorème 6. *Le problème* CSP_{bool} *n'admet pas de noyau polynomial paramétré par l'ensemble de paramètres* $\{\boldsymbol{pw}, \boldsymbol{pw}^d, \boldsymbol{dep}\}$*, sauf si* $\mathcal{NP} \subseteq \mathrm{co}\mathcal{NP}/\mathrm{poly}$.

Afin de démontrer ce résultat, on prouve que la OU-composition utilisée par Gaspers et Szeider pour le problème $\mathrm{CSP}_{\mathrm{bool}}$ paramétré par **tw** est également une OU-composition pour le problème $\mathrm{CSP}_{\mathrm{bool}}$ paramétré par $\{\mathbf{pw}, \mathbf{pw}^d, \boldsymbol{dep}\}$.

Démonstration. Soit $I_1, \ldots, I_t$ une séquence d'instances du problème $\mathrm{CSP}_{\mathrm{bool}}$ où $I_i = (\mathcal{V}_i, \{0,1\}, \mathcal{C}_i)$ pour $i \in [1,t]$, s.p.d.g. supposons que pour tout $1 \leq i < j \leq t$, $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$.

Notons respectivement G_i et G_i^d le graphe primal et le graphe dual de l'instance I.

De plus notons respectivement P_i une décomposition en chemin du graphe G_i et P_i^d une chemin décomposition du graphe G_i^d.

Construisons l'instance $I = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ à partir des instances $I_1, \ldots, I_t$ telle que :

- $\mathcal{V} = \bigcup_{i=1}^{t} \mathcal{V}_i \cup \{a_1, \ldots, a_t, b_0, \ldots, b_t\}$,
- Toutes les variables ont pour domaine $\{0,1\}$ sauf b_0 dont le domaine est $\mathcal{D}(b_0) = \{0\}$ et b_t dont le domaine est $\mathcal{D}(b_t) = \{1\}$.
- Les contraintes de I sont l'union des contraintes construites ci-dessous :
  - Pour chaque $i \in [1,t]$ et pour chaque contrainte C de $\mathcal{C}_i$ telle que $\mathit{scope}(C) = (x_1, \ldots, x_r)$, on crée une contrainte C' telle que $\mathit{scope}(C') = (a_i, x_1, \ldots, x_r)$ et telle que :

$$\mathit{list}(C') = \{(0, v_1, \ldots, v_r) \,:\, (v_1, \ldots, v_r) \in \mathit{list}(C)\} \cup \{(1,1,\ldots,1)\}$$

  - Pour chaque $i \in [1,t]$, on crée une contrainte ternaire C_i^* telle que $\mathit{scope}(C_i^*) = (b_{i-1}, b_i, a_i)$ et telle que $\mathit{list}(C_i^*) = \{(0,0,1), (0,1,0), (1,1,1)\}$,

Remarquons que les contraintes C^0, C^1 et C_i^* sont satisfaites par une assignation α ssi il existe un indice $1 \leq i \leq t$ tel que :

- Pour tout $j < i$, $\alpha(b_j) = 0$ et pour tout $k \geq i$, $\alpha(b_k) = 1$,
- Pour tout $i \neq j$, $\alpha(a_j) = 1$ et $\alpha(a_i) = 0$.

Si il existe une assignation α satisfaisant I alors les contraintes C_j^*, C^0 et C^1 sont satisfaites.

Par la remarque formulée ci-dessus, cela n'est possible que si il existe $i \in [1,t]$ tel que $\alpha(a_i) = 0$ or si c'est le cas par construction, la restriction de α à $\mathcal{V}_i$ satisfait toutes les contraintes de l'instance I_i.

Inversement, si il existe un indice i tel que I_i soit consistante par une assignation α alors I est consistante avec l'assignation β définie ci-dessous :

$$\beta(v) = \begin{cases} \alpha(v) \text{ si } v \in \mathcal{V}_i \\ 0 \text{ si } v \in \{a_i\} \cup \{b_j \,:\, j < i\} \\ 1 \text{ sinon.} \end{cases}$$

Donc il existe une assignation satisfaisant I ssi il existe un indice $i \in [1, t]$ tel qu'il existe une assignation satisfaisant l'instance I_i

Prouvons que les paramètres **pw**, $\mathbf{pw}^d$ et ***dep*** de l'instance I sont polynomialement bornés par les paramètres **pw**, $\mathbf{pw}^d$ et ***dep*** des instances I_i.

Les figures suivantes représentent le graphe primal et le graphe dual de I pour $t = 4$.

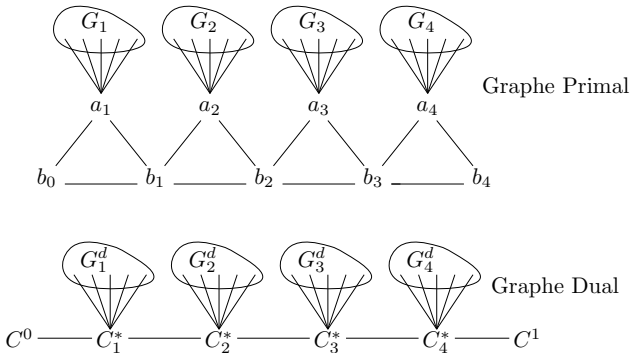

FIGURE 3.3: Graphe primal et dual de I pour $t = 4$

- On peut obtenir une décomposition en chemin du graphe primal de l'instance I à partir des décompositions en chemins P_i.

  En augmentant la largeur de deux comme l'explique la figure 3.4 pour $t = 4$.

  Dans cette figure $P_i + a_i + b_i$ schématise la décomposition en chemin P_i où on ajoute les sommets a_i et b_i dans les sacs associés à chaque nœud de la décomposition en chemin P_i.

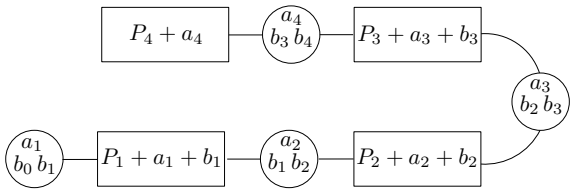

FIGURE 3.4: Décomposition arborescente du graphe primal de I

- De la même manière on peut obtenir une décomposition en chemin du graphe dual à partir des décompositions en chemin P_i^d en augmentant la largeur d'une unité.

- Le nombre de tuples maximum d'une contrainte de I est égal à celui des instances I_i plus une unité par construction.

Nous avons prouvé que :

- il existe une assignation satisfaisant I ssi il existe un indice $i \in [1, t]$, tel qu'il existe une assignation satisfaisant l'instance I_i,

- Les paramètres **pw**, $\mathbf{pw}^d$ et ***dep*** de l'instance I sont polynomialement bornés par les paramètres **pw**, $\mathbf{pw}^d$ et ***dep*** des instances I_i.

La transformation des instances I_i en l'instance I est bien polynomiale en $\sum_{i=1}^{t} |I_i| + k$, donc il s'agit bien d'une OU-composition pour le problème CSP_{bool} paramétré par l'ensemble de paramètres $\{\mathbf{pw}, \mathbf{pw}^d, \boldsymbol{dep}\}$. □

3.3 Autres paramètres graphiques

Les deux théorèmes suivants traitent de paramètres qui dominent la largeur de chemin.

Ils s'appuient tous les deux sur une transformation paramétrée polynomiale simple du problème k-Coloration au problème CSP.

Avant de décrire cette transformation, rappelons qu'une instance du problème k-Coloration se compose d'un graphe $G = (V, E)$ et d'un entier $k > 0$, le but étant de savoir si le graphe G est coloriable avec k couleurs.

Lemme 1. *Il existe une transformation paramétrée polynomiale du problème k-*Coloration *paramétré par le nombre de couleurs vers le problème 2-*CSP *paramétré par* ***dom*** *et* ***dep****.*

*Cette transformation transforme un graphe G d'une instance de k-*Coloration *en une instance I de 2-*CSP *telle que le graphe primal de I soit identique à G.*

Démonstration. Soit une instance du problème de la k-Coloration (G, k) où $G = (\mathcal{V}, E)$.

On lui associe l'instance $I = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ du problème 2-CSP telle que :

- $\mathcal{D}(x) = \{1, \ldots, k\}$ pour tout $x \in \mathcal{V}$,
- $\mathcal{C} = \bigcup_{e \in E} C_e$. Pour toute arête $e = (u, v)$, $\mathit{scope}(C_e) = (u, v)$ et $\mathit{list}(C_e) = \{(i, j) : i, j \in [1, k],\ i \neq j\}$.

Trivialement, le graphe G est k-coloriable ssi I est satisfiable. Les paramètres ***dom*** et ***dep*** sont bien polynomialement bornés par k : $\boldsymbol{dom} = k$ et $\boldsymbol{dep} = k(k-1)$.

Deux variables correspondent à des sommets adjacents dans le graphe primal ssi elles apparaissent dans la même contraintes or c'est le cas ssi elles correspondent à deux sommets adjacents dans G.

Donc le graphe G et le graphe primal de I sont bien identiques. Ainsi les paramètres du graphe G sont identiques à ceux du graphe primal de I comme par exemple la taille d'un vertex cover minimum. □

Le théorème suivant énonce deux résultats de la thèse de Bart Maarten Paul Jansen [26] transposable au problème CSP grâce à la transformation décrite ci-dessus.

Un *linear feedback vertex set* est un ensemble de sommet dont l'exclusion transforme le graphe en une union de chemins disjoints.

Théorème 7 ([26])**.** *Sauf si $\mathcal{NP} \subseteq \text{co}\mathcal{NP}/\text{poly}$:*

*1) Le problème de la k-*Coloration *paramétré par le nombre de couleurs et la taille d'un vertex cover minimum n'admet pas de noyau polynomial,*

*2) Le problème de la 3-*Coloration *paramétré par la taille d'un linear feedback vertex set minimum n'admet pas de noyau polynomial.*

Le corolaire suivant est une conséquence du théorème 7 et 2 ainsi que du lemme 1.

Corollaire 1. *Sauf si* $\mathcal{NP} \subseteq$ co$\mathcal{NP}$/poly *:*

1. *Le problème 2-*CSP *paramétré par la taille d'un vertex cover minimum du graphe primal,* ***dom*** *et* ***dep*** *n'admet pas de noyau polynomial,*

2. *Le problème 2-*CSP *paramétré par la taille d'un linear feedback vertex set minimum du graphe primal n'admet pas de noyau polynomial même quand* $\boldsymbol{dom} = 3$ *et* $\boldsymbol{dep} = 6$*.*

3.4 Nombre de variables, nombre de contraintes et taille du domaine

En prenant pour paramètre le nombre de variables, le nombre de contraintes et la taille du domaine, on capture trois des quatre dimensions d'une instance de CSP[2]

Pourtant même avec ces trois paramètres, ce problème n'admet toujours pas de noyau polynomial comme l'annonce le théorème ci-dessous et ce même quand on fixe ***dom*** à 2.

Théorème 8. *Le problème* CSP_{bool} *paramétré par l'ensemble de paramètres* $\{\boldsymbol{vars}, \boldsymbol{cons}\}$ *n'admet pas de noyau polynomial.*

L'ensemble de paramètres $\{\boldsymbol{vars}, \boldsymbol{dom}, \boldsymbol{cons}\}$ domine tous les paramètres présentés dans ce mémoire sauf ***dep***, celui ci ne peut être majoré polynomialement par ***vars***, ***dom*** et ***cons***.

Démonstration. Pour prouver ce résultat démontrons qu'il existe une composition croisée du problème Clique vers le problème CSP_{bool} paramétré par $\{\boldsymbol{vars}, \boldsymbol{cons}\}$.

Soit la relation d'équivalence Θ définie sur les instances du problème Clique telles que $(G_1, k_1)\Theta(G_2, k_2)$ ssi G_1 et G_2 possèdent le même nombre de sommets et $k_1 = k_2$, la relation Θ est une relation d'équivalence polynomiale.

Soit $((G_1, k), \ldots, (G_t, k))$ une séquence d'instances de Clique deux à deux équivalentes avec la relation Θ, soit V_i l'ensemble de sommets du graphe G_i et E_i son ensemble d'arêtes. Supposons s.p.d.g. que $V_i = \{1, \ldots, n\}$ pour $1 \leq i \leq t$.

Soit $c = \lceil \log_2(t) \rceil$ et Ψ une fonction de $[1, t] \times [1, n] \times [1, n]$ vers $\{0, 1\}^{c+2n}$ dont la définition est dans expliquée dans la figure ci-dessous :

Soit l'ensemble de tuples $R = \bigcup_{z=1}^{t} \{\Psi(z, u, v), \Psi(z, v, u) : (u, v) \in E_z\}$.

Soit l'instance $I = (\mathcal{V}, \{0, 1\}, \mathcal{C})$ du problème CSP_{bool} construite à partir des instances de cliques $((G_1, k), \ldots, (G_t, k))$ telle que :

- $\mathcal{V} = \{Y_1, \ldots, Y_c\} \bigcup_{i=1}^{k} \{X_i^1, X_i^2, \ldots, X_i^n\}$,

[2]La quatrième dimension étant la profondeur : ***dep***.

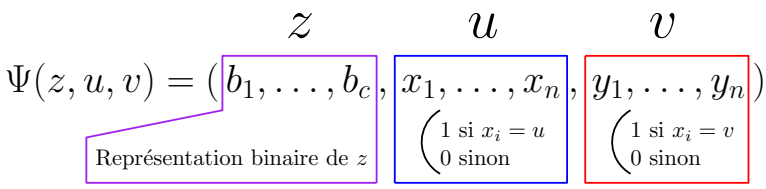

FIGURE 3.5: Définition de Ψ.

- $\mathcal{C} = \bigcup_{1 \leq i < j \leq k} C_{ij}$. Pour tout $1 \leq i < j \leq k$:

$$scope(C_{ij}) = (Y_1, \ldots, Y_c, X_i^1, \ldots, X_i^n, X_j^1, \ldots, X_j^n) \text{ et } list(C_{ij}) = R$$

Prouvons qu'il existe une assignation satisfaisant I ssi il existe un indice λ tel qu'il existe une clique de taille k dans G_λ :

- Supposons qu'il existe un indice $\lambda \in [1, t]$ tel qu'il existe un ensemble de sommets $\{x_1^*, \ldots, x_k^*\}$ formant une clique de taille k dans G_λ.

  Soit l'assignation α telle que :

  - Pour tout $i \in [1, k]$, $j \in [1, n]$, $\alpha(X_i^j) = 1$ si $j = x_i^*$ sinon $\alpha(X_i^j) = 0$.
  - Pour tout $i \in [1, c]$, $\alpha(Y_i) = b_i$ tel que $b_1 b_2 \cdots_c$ soit la représentation binaire de λ sur c bits.

  L'assignation α satisfait toutes les contraintes car pour tout $i \in [1, k]$, $j \in [1, n]$, (x_i^*, x_j^*) est une arête de G_i donc le tuple $\Psi(\lambda, x_i^*, x_i^*)$ est un tuple de R.

- Si I est consistante alors soit α une assignation des variables de $\mathcal{V}$ satisfaisant I. Par construction, $\alpha(Y_1)\alpha(Y_2)\ldots\alpha(Y_c)$ est la représentation binaire d'un nombre λ entre 1 et t. Et pour chaque $i \in [1, k]$, il existe un unique $j \in [1, n]$ tel que $\alpha(X_i^j) = 1$.

  Soit $\{x_1^*, \ldots, x_k^*\} \subseteq \{1, \ldots, n\}$ tel que pour tout $i \in [1, k]$, $X_i^{x_1^*} = 1$.

  Soit $1 \leq i < j \leq k$: la contrainte C_{ij} est satisfaite par α donc $\Psi(\lambda, x_i^*, x_j^*) \in R$ ce qui implique que (x_i^*, x_j^*) est une arête de G_λ. Donc $\{x_1^*, \ldots, x_k^*\}$ est une clique de G_λ.

La transformation de la séquence $((G_1, k), \ldots, (G_t, k))$ en l'instance I est effectuable en temps polynomial en $\sum_{i=1}^t |G_i|$.

De plus $\boldsymbol{vars} = n + \lceil \log_2(t) \rceil$ et $\boldsymbol{cons} = k(k-1)/2 \leq n^2$, donc cette transformation est bien une composition croisée du problème CLIQUE vers le problème CSP_{bool} paramétré par $\boldsymbol{vars}$ et $\boldsymbol{cons}$. Le théorème 4 permet de conclure. □

3.5 Discussion

Voici une synthèse des résultats que nous avons trouvé concernant le CSP où les contraintes sont représentées par une liste de tuples autorisés :

Sauf si $\mathcal{NP} \subseteq \text{co}\mathcal{NP}/\text{POLY}$, les problèmes paramétrés suivant n'admettent pas de noyau polynomial :

1) CSP_{bool} paramétré par $\mathbf{pw}$, $\mathbf{pw}^d$ et $\boldsymbol{dep}$,

2) 2-CSP paramétré par $\boldsymbol{dom}$, $\boldsymbol{dep}$ et la taille d'un vertex cover minimum du graphe primal,

3) 2-CSP paramétré par la taille d'un linear feedback vertex set minimum du graphe primal même quand $\boldsymbol{dom} = 3$ et $\boldsymbol{dep} = 6$,

4) CSP_{bool} paramétré par $\boldsymbol{vars}$ et $\boldsymbol{cons}$.

Ces résultats renforcent l'intuition que le problème CSP et les problèmes en IA en général sont des problèmes résistants à la kernelization polynomiale.

Ces résultats dévoilent les limites des méthodes de preprocessing très utilisées dans les solveurs CSP actuels.

Peu importe la méthode de preprocessing, si elle est polynomiale, elle ne peux vraisemblablement pas être formalisée comme une kernelization polynomiale du problème CSP paramétré par un ensemble de paramètres dominés par $\{\boldsymbol{vars}, \boldsymbol{cons}, \boldsymbol{dom}\}$.

Il serait intéressant de d'avoir des bornes inférieures pour d'autre ensemble de paramètres comme $\{\boldsymbol{cons}, \boldsymbol{dom}, \boldsymbol{dep}\}$ ou encore $\{\boldsymbol{vars}, \boldsymbol{dom}, \boldsymbol{dep}\}$.

Le but serait d'utiliser la hiérarchie de paramètres pour obtenir une caractérisation complète des paramétrisations du CSP par rapport à la kernelization.

Enfin, il serait intéressant d'étendre ces résultats à une notion plus forte que la kernelization qui est la Turing Kernelization dont la définition est ci-dessous.

Définition 8 (Turing kernelization)**.** *Un problème paramétré $L \subseteq \Sigma^* \times \mathbb{N}$ admet une Turing kernelization ssi il existe une fonction calculable g et un algorithme qui pour toute instance (x, k) de L, décide en temps polynomial si $(x, k) \in L$ en utilisant un oracle pour l'ensemble :*

$$\{(x', k') : (|x'| \leq g(k)) \wedge (k' \leq g'(k)) \wedge (x', k') \in L)\}$$

g est la taille du Turing kernel. Si g est un polynôme alors on parle de Turing kernel polynomial.

Certains problèmes paramétrés n'admettent pas de noyau polynomiaux mais admettent un Turing kernel polynomial [34]. Similairement à la W-hiérarchie pour la notion de FPT, Hermelin et al. ont introduit une hiérarchie pour la kernelization prenant en compte la notion de Turing kernelization [23, 24].

Il serait intéressant de connaitre la position des problèmes paramétrés étudiés dans ce mémoire dans cette hiérarchie.

4

Contraintes Globales

4.1 Définitions et exemples

La représentation des contraintes par une liste de tuples autorisés n'est pas toujours satisfaisante. Quand les tuples autorisés peuvent être caractérisés logiquement ce qui est fréquent, il est préférable de donner une définition implicite de la contrainte. On appelle ces contraintes des contraintes globales, il en existe des centaines.

Par exemple, il est souvent souhaité qu'un ensemble de variables prennent toutes des valeurs différentes pour modéliser un problème d'emplois du temps ou un problème d'ordonnancement.

Ainsi la plupart des solveurs CSP intègre la contrainte globale AllDifferent$(X_1, \ldots, X_n)$ qui est satisfaite par une assignation ssi celle-ci assigne n valeurs différentes aux variables $X_1, \ldots, X_n$.

Comme dans le problème CSP chaque variable dans la portée de la contrainte possède un domaine noté $\mathcal{D}(X_i)$ et une assignation des variables est une fonction assignant à chaque variable une valeur de son domaine.

Un des problèmes les plus étudiés concernant les contraintes globales consiste à savoir pour une contrainte donnée si il existe une assignation la satisfaisant. Si une telle assignation existe, on dit que la contrainte en question est consistante.

Notons Cons-Ψ le problème consistant à savoir si la contrainte Ψ est consistante.

Le problème Cons-AllDifferent est polynomial [32] mais Cons-Ψ est $\mathcal{NP}$-complet pour de nombreuses contraintes globales [4].

Certaines des contraintes globales étudiés dans ce mémoire utilisent des ensembles-variables. Ce type de variables est définie par un ensemble de valeurs obligatoires et un ensemble de valeurs facultatives. Pour un ensemble variable S, notons $lb(S)$ le premier ensemble et $ub(S)$ le dernier, une assignation α vérifie $lb(S) \subseteq \alpha(S) \subseteq ub(S)$.

Définissons les contraintes globales qui nous intéressent :

- Among$(X_1, \ldots, X_n, S, N)$ est une contrainte où S est un ensemble-variable et où N est une variable à valeurs entières. Cette contrainte est satisfaite par une assignation α ssi $\alpha(N) = |\{X_i : \alpha(X_i) \in \alpha(S)\}|$.

- Roots$(X_1, \ldots, X_n, S, T)$ est une contrainte où S et T sont des ensembles-variables. Cette contrainte est satisfaite par une assignation α ssi $\alpha(S) = \{i : \alpha(X_i) \in \alpha(T)\}$.

- COMMON$(X_1, \ldots, X_n, Y_1, \ldots, Y_m, N, M)$ est une contrainte constitué de deux ensembles de variables $X = \{X_1, \ldots, X_n\}$ et $Y = \{Y_1, \ldots, Y_m\}$ et de deux variables à valeurs entières N et M. Cette contrainte est consistante ssi il existe une assignation α telle que $\alpha(N) = |\{X_i : \alpha(X_i) \in \alpha(Y)\}|$ et $\alpha(M) = |\{Y_j : \alpha(Y_j) \in \alpha(X)\}|$.

Les problèmes CONS-AMONG, CONS-ROOTS et CONS-COMMON sont tous $\mathcal{NP}$-complets [1, 2, 5]. Cependant l'article [3] démontre que CONS-AMONG et CONS-ROOTS sont FPT en prenant pour paramètre $|ub(S)\backslash lb(S)|$ pour CONS-AMONG et $|ub(T)\backslash lb(T)|$ pour CONS-ROOTS.

4.2 Algorithme paramétré pour Common

Le théorème suivant décrit une paramétrisation du problème CONS-COMMON telle que ce problème soit FPT.

Théorème 9. *Le problème* CONS-COMMON *paramétré par* $k = |\mathcal{D}(X) \cap \mathcal{D}(Y)|$ *est FPT.*

Démonstration. Quelques notations avant de commencer : Soit v une valeur de $\mathcal{D}(X) \cup \mathcal{D}(Y)$, soit α une assignation des variables, notons $\alpha^{-1}(v) = \{Z \in X \cup Y : \alpha(Z) = v\}$. Soit f une fonction partant d'un ensemble E, soit $E' \subseteq E$, notons $f(E) = \bigcup_{e \in E} f(e)$.

Soit COMMON$(X_1, \ldots, X_n, Y_1, \ldots, Y_m, N, M)$ une instance de CONS-COMMON. Notons $B = \mathcal{D}(X) \cap \mathcal{D}(Y)$ et $U = \mathcal{D}(X) \cup \mathcal{D}(Y)$.

L'idée de l'algorithme FPT est de regarder pour chaque partition possible de B en trois ensembles I, P_X et P_Y si il existe une assignation α telle que I soit l'ensemble des valeurs assignées à la fois à une variable de X et une variable de Y et telle qu'aucune valeur de P_X (resp. P_Y) est assignée à une variable de Y (resp. de X). De plus, le nombre de variables de X (resp. Y) assignées à une valeur de I doit être égal à $\alpha(N)$ (resp. $\alpha(M)$).

Formellement, ces propriétés sont les suivantes :

(i) $I \subseteq \alpha(Y) \subseteq U\backslash P_X$,

(ii) $|\alpha^{-1}(I) \cap Y| = \alpha(M)$,

(iii) $I \subseteq \alpha(X) \subseteq U\backslash P_Y$,

(iv) $|\alpha^{-1}(I) \cap X| = \alpha(N)$.

Les deux premières propriétés sont symétriques aux deux dernières, de plus si il existe une assignation vérifiant (i) et (ii) et une autre vérifiant (iii) et (iv) alors il existe une assignation vérifiant les quatre propriétés. Ainsi, dans la suite de cette preuve, nous verrons seulement comment vérifier si il existe une assignation vérifiant (i) et (ii).

Fixons la partition I, P_X, P_Y et supposons qu'il n'existe pas de $y \in Y$ tel que $\mathcal{D}(y) \subseteq P_X$ sinon il n'existerait pas d'assignation vérifiant (i).

Notons $\tilde{Y} = \{Y_i : \mathcal{D}(Y_i) \subseteq I\}$, c'est l'ensemble des variables « obligatoires », ne pouvant prendre que des valeurs de I.

Notons $\overline{Y} = \{Y : \mathcal{D}(Y) \cap I \neq \varnothing\}\backslash\tilde{Y}$, c'est l'ensemble des variables « facultatives » pouvant prendre ou non une valeur de I.

Supposons s.p.d.g. que $Y = \tilde{Y} \cup \overline{Y}$ les variables de Y ne pouvant pas être assignée à des valeurs de I n'interviennent pas dans l'existence d'une assignation vérifiant les deux propriétés.

Soit l'ensemble E des entiers δ tels qu'il existe une assignation α vérifiant (i) telle que $\delta = |\alpha^{-1}(I) \cap Y|$.

Remarquons que E est un intervalle, supposons qu'il ne soit pas vide alors il existe une assignation α telle que $|\alpha^{-1}(I) \cap Y|$ soit minimum.

Si $\overline{Y} \backslash \alpha^{-1}(I)$ est vide alors $|\alpha^{-1}(I) \cap Y|$ est maximum (toutes les variables pouvant être assignés à une valeur de I l'étant). Si ce n'est pas le cas, nous pouvons modifier α en assignant une variable de $\overline{Y} \backslash \alpha^{-1}(I)$ à une valeur de I et ainsi augmenter d'une unité la valeur de $|\alpha^{-1}(I) \cap Y|$.

Nous pouvons faire cette opération jusqu'à ce que $\overline{Y} \backslash \alpha^{-1}(I)$ soit vide et donc jusqu'à atteindre la borne supérieure de E.

De cette remarque nous pouvons affirmer qu'il existe une assignation vérifiant les propriétés (i) et (ii) ssi $E \cap \mathcal{D}(M) \neq \varnothing$. Si E n'est pas vide, sa borne supérieure est égale à $|\tilde{Y} \cup \overline{Y}|$

Donc pour calculer E, il suffit de savoir comment calculer sa borne inférieure.

Pour calculer la borne inférieure de E, nous utilisons une variante du problème du flot maximum où à chaque arc est associé une capacité mais aussi un débit minimum. Tout flot valide devant faire passer par chaque arc un flot supérieur à son débit minimum et inférieur à sa capacité.

L'instance de ce problème de flot qui nous intéresse est décrite dans la figure suivante :

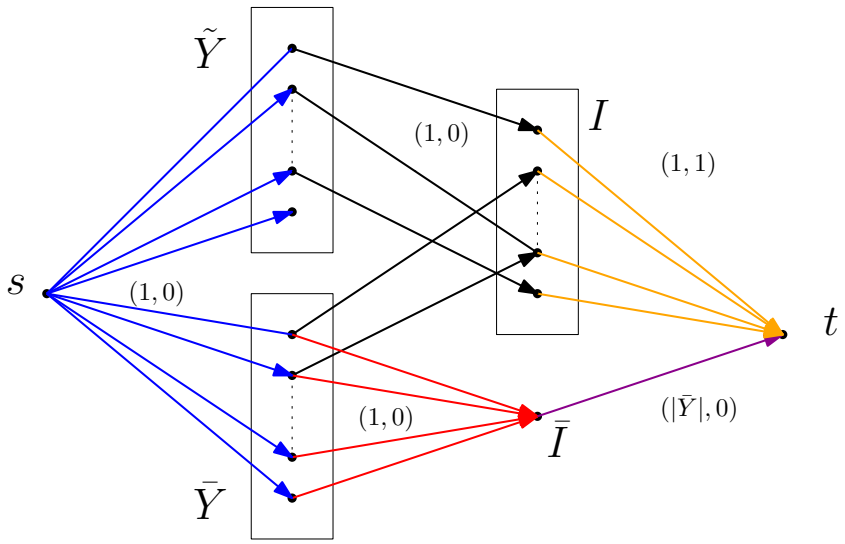

FIGURE 4.1: Instance du problème de flot en question

Plus formellement, cette instance est composée d'un graphe orienté $G = (V, A)$ où :

- $V = \{s, t, \bar{I}\} \cup Y \cup I$,
- $A = E_1 \cup E_2 \cup E_3 \cup E_4 \cup \{(\bar{I}, t)\}$ où :

- $E_1 = \{(s, y) : y \in Y\}$, les arcs en bleu dans la figure,
- $E_2 = \{(y, v) : y \in Y, v \in I \cap \mathcal{D}(y)\}$, les arcs en noir dans la figure,
- $E_3 = \{(y, \bar{I}) : y \in \overline{Y}\}$, les arcs en rouge dans la figure,
- $E_4 = \{(v, t) : v \in I\}$, les arcs en orange dans la figure.

- Tous les arcs ont une capacité unitaire sauf l'arc (en violet dans la figure) $(\bar{I}, t)$ qui possède une capacité égale à $|\overline{Y}|$,
- Tous les arcs ont un flot minimum nul sauf les arcs appartenant à E_4 qui ont un débit minimum unitaire.

Remarquons que :

- À partir d'une assignation α vérifiant (i), on peut construire un flot f tel que $|\alpha^{-1}(U \backslash I) \cap \overline{Y}| = f(\overline{Y}, \bar{I})$., voici les étapes de construction :

  Pour chaque valeur v de I il existe une variable $x \in \alpha^{-1}(v)$, car α vérifie (i). Donc pour chaque valeur $v \in I$ nous pouvons augmenter le flot du chemin $s - x - v - t$ d'une unité. Après cette opération, pour chaque $v \in I$, on a $f(v, t) = 1$ donc le flot f est bien valide.

  Pour chaque variable $y \in \overline{Y}$ telle que $\alpha(y) \notin I$ augmentons le flot du chemin $s - y - \bar{I} - t$ d'une unité. Après cette étape, le flot f vérifie l'égalité $|\alpha^{-1}(U \backslash I) \cap \overline{Y}| = f(\overline{Y}, \bar{I})$.

- Inversement à partir d'un flot f, il est possible de construire une assignation α vérifiant $|\alpha^{-1}(U \backslash I) \cap \overline{Y}| \geq f(\overline{Y}, \bar{I})$.

  Pour chaque valeur $v \in I$, on a $f(v, t) = 1$ donc il existe une variable $x \in Y$ telle que $f(x, v) = 1$, assignons v à x, à la fin de cette étape l'assignation vérifie (i).

  Pour les autres variables $y \in Y$ n'étant pas assignées lors de cette étape, si $y \in \tilde{Y}$ assignons à y une des valeurs de son domaine. Sinon si $y \in \overline{Y}$ assignons à y une valeur de son domaine n'appartenant pas à I. Ainsi l'assignation vérifie $|\alpha^{-1}(U \backslash I) \cap \overline{Y}| \geq f(\overline{Y}, \bar{I})$.

- Une assignation α vérifiant (i) et telle que $|\alpha^{-1}(I) \cap Y|$ soit minimum, maximise $|\alpha^{-1}(U \backslash I) \cap \overline{Y}|$ car seules les variables de $\overline{Y}$ peuvent prendre des valeurs n'appartenant pas à I. De même, un flot maximum f maximise $f(\overline{Y}, \bar{I})$, tout flot g valide a pour valeur $g(\overline{Y}, \bar{I}) + |I|$.

De ces trois remarques nous pouvons affirmer que :

- Il existe une assignation vérifiant (i) ssi il existe un flot valide dans G,
- soit f un flot valide de G et F sa valeur, si f est un flot maximum alors $\inf(E) = |\tilde{Y}| + |\overline{Y}| - F + |I|$.

Prouvons que la contrainte Common$(X_1, \ldots, X_n, Y_1, \ldots, Y_m, N, M)$ est consistante ssi il existe une partition I, P_X, P_Y de B telle qu'il existe une assignation vérifiant les quatre propriétés (i),(ii),(iii) et (iv).

- Supposons qu'il existe une assignation α satisfaisant la contrainte (i) alors $I = \alpha(Y) \cap \alpha(X)$.

  Par définition $I \subseteq B$. Soit $P_X = (B \backslash I) \cap \alpha(Y)$ et $P_Y = B \backslash (I \cup P_X)$. Les ensembles I, P_X, P_Y forment bien une partition de B. L'assignation α vérifie bien les propriétés (i) et (iii) avec cette partition de B. On a $\alpha(N) = |\{X_i : \alpha(X_i) \in \alpha(Y)\}|$ donc $\alpha(N) = |\{X_i : X_i \in I\}|$, comme I est l'ensemble des valeurs assignées à la fois à une variable de X et de Y. Donc α vérifie la propriété (iv), symétriquement nous pouvons prouver qu'elle vérifie la propriété (ii).

- Supposons qu'il existe une assignation α vérifiant les quatre propriétés pour une partition I, P_X, P_Y de B donnée.

  Les propriétés (i) et (iii) impliquent que $\alpha(Y) \cap \alpha(X) = I$. Or $|\alpha^{-1}(I) \cap X| = \alpha(N)$ donc $\alpha(N) = |\{X_i : \alpha(X_i) \in \alpha(Y)\}|$. Symétriquement on peut montrer que $\alpha(M) = |\{Y_j : \alpha(Y_j) \in \alpha(Y)\}|$.

Il existe 3^k partitions possibles de B en trois ensembles ($k = |B| = |\mathcal{D}(X) \cap \mathcal{D}(Y)|$).

Pour chaque partition, notre algorithme vérifie en temps polynomial si il existe une assignation vérifiant les quatre propriétés donc cet algorithme possède un temps d'exécution en $O^*(3^k)$. Le problème Cons-Common est bien FPT pour le paramètre $|\mathcal{D}(X) \cap \mathcal{D}(Y)|$. □

4.3 Kernelization et implications d'ETH

Le théorème suivant affirme que les trois problèmes paramétrés abordés ci-dessus n'admettent vraisemblablement pas de noyau polynomial.

En réalité, il affirme un résultat un peu plus fort en affirmant qu'il n'existe pas de noyau polynomial pour ces trois problèmes quand ils sont paramétrés par ***dom***. Ici ***dom*** est le cardinal de l'union des domaines des variables et des ensembles de valeurs facultatives des ensembles-variables.

Théorème 10. *Sauf si* $\mathcal{NP} \subseteq$ co$\mathcal{NP}$/poly, *les problèmes* Cons-Among, Cons-Roots *et* Cons-Common *n'admettent pas de noyau polynomial quand ils sont paramétrés par* ***dom***.

La preuve de ce théorème est basée sur les preuves présentées dans l'article [20] et celles de la $\mathcal{NP}$-complétude des problèmes Cons-Among et Cons-Roots dans les articles [1, 2].

Démonstration. Prouvons ces trois résultats avec trois transformations paramétrées toutes les trois depuis le problème SAT paramétré par le nombre de variables.

Soit $F = \{C_1, \ldots, C_m\}$ un ensemble de clauses sur les variables $x_1, \ldots, x_n$.

- Soit $k = n + m + 1$, associons à cette instance de SAT une instance de Cons-Among, commençons par présenter les variables et leurs domaines :

  – À chaque variable x_i de F, on associe $2k + 1$ variables $X_i^1, \ldots, X_i^{2k+1}$ telles que :

  - $\mathcal{D}(X_i^j) = \{x_i\}$ pour $j \in [1, k]$,
  - $\mathcal{D}(X_i^j) = \{\neg x_i\}$ pour $j \in [k+1, 2k]$,
  - $\mathcal{D}(X_i^{2k+1}) = \{x_i, \neg x_i\}$.
- À chaque clauses C_j, on associe une variable Y_j dont le domaine est $\mathcal{D}(Y_j) = \{l : l \in C_j\}$.
- La variable N n'admet dans son domaine que la valeur $n(k+1)+m$.
- L'ensemble-variable S est défini par $lb(S) = \varnothing$ et $ub(S) = \{x_i, \neg x_i : 1 \leq i \leq n\}$.

F est satisfiable ssi la contrainte $\textsc{Among}(X_1^1, X_1^2, \ldots, X_n^{2k}, X_n^{2k+1}, Y_1, \ldots, Y_m, N, S)$ est consistante :

Si F est satisfiable, il existe une interprétation I satisfaisant F. À partir de I, on peut construire une assignation α satisfaisant la contrainte Among, α vérifie :

- Pour tout $i \in [1, n]$, $\alpha(X_i^{2k+1}) = x_i$ si $I(x_i) = \text{true}$ sinon $\alpha(X_i^{2k+1}) = \neg x_i$,
- Pour tout $j \in [1, m]$, $\alpha(Y_m) = l$ où l est un littéral interprété à vrai par I pour la clause C_j,
- $\alpha(Y) = \{\alpha(X_1^{2k+1}, \ldots, X_n^{2k+1}\}$.

L'autre sens est trop laborieux pour figurer dans ce mémoire, il requiert de prouver dans un premier temps que toute assignation α satisfaisant la contrainte vérifie $|\alpha(S) \cap \{x_i, \neg x_i\}| = 1$ pour tout $1 \leq i \leq n$. Une fois prouvée, cette remarque permet de facilement associer une interprétation satisfaisant F à partir d'une assignation satisfaisant la contrainte.

- Soit $\textsc{Roots}(X_1, \ldots, X_n, Y_1, \ldots, Y_m, S', T)$ l'instance de Cons-Roots associée à F où :
  - Pour tout $1 \leq i \leq n$, $\mathcal{D}(X_i) = \{x_i, \neg x_i\}$,
  - Pour tout $1 \leq j \leq m$, $\mathcal{D}(Y_j) = \{l : l \in C_j\}$,
  - L'ensemble-variable S' est fixé à la valeur $\{1, \ldots, n\}$. Il force ainsi les n premières variables $X_1, \ldots, X_n$ à prendre des valeurs dans T tandis que les variables $Y_1, \ldots, Y_m$ ne peuvent pas prendre de valeurs dans T.
  - L'ensemble-variable T est définie par $lb(T) = \varnothing$ et $ub(T) = \{x_i, \neg x_i : 1 \leq i \leq n\}$.

  F est satisfiable ssi la contrainte $\textsc{Roots}(X_1, \ldots, X_n, Y_1, \ldots, Y_m, S', T)$ admet une assignation la satisfaisant. Le sens direct est simple. Pour l'autre sens, il suffit de remarquer qu'une assignation α satisfaisant la contrainte Roots vérifie que pour chaque variable Y_j, il existe un littéral l dans la clause C_j telle que $\alpha(x) = \neg l$ où x est la variable associée à l.

  Avec cette remarque, on peut prouver que l'interprétation β des variables de F telles que $\beta(x_i) = 1$ ssi $\alpha(X_i) = \neg x_i$ satisfait chaque clause.

- Soit Common$(X_1, \ldots, X_n, Y_1, \ldots, Y_m, N, M)$ l'instance de Cons-Common associée à F où :
  - Pour tout $1 \leq i \leq n$, $\mathcal{D}(X_i) = \{x_i, \neg x_i\}$,
  - Pour tout $1 \leq j \leq m$, $\mathcal{D}(Y_j) = \{l : l \in C_j\}$,
  - $\mathcal{D}(N) = \{1, \ldots, n\}$ et $\mathcal{D}(M) = \{m\}$.

  F est satisfiable ssi la contrainte Common$(X_1, \ldots, X_n, Y_1, \ldots, Y_m, N, M)$ est consistante. Ce résultat est simple à prouver.

Les temps d'exécution et la taille des instances obtenues de ces trois transformations sont bien polynomiaux en $n + m$.

Les instances sont bien équivalentes et les paramètres des instances obtenues sont bien polynomialement bornés par n : pour la première ***dom*** est égal à $2n$, pour la deuxième ***dom*** est égal à $3n$ et pour la troisième, il est égal à $3n + 1$.

Il s'agit bien de transformations paramétrées polynomiales depuis SAT paramétré par le nombre de variables, or ce problème n'admet de pas noyau polynomial sauf si $\mathcal{NP} \subseteq$ co$\mathcal{NP}$/poly. Le théorème 2 permet de conclure. □

Théorème 11. *Sauf si ETH est fausse, les problèmes* Cons-Among, Cons-Roots *et* Cons-Common *paramétrés par* ***dom*** *n'admettent pas d'algorithmes sous-exponentiels, i.e. en temps* $O(2^{o(\boldsymbol{dom})})$.

Démonstration. Les trois transformations paramétrées polynomiales décrites plus haut sont des transformations linéaires : le paramètre de l'instance d'arrivée est linéaire par rapport au paramètre de l'instance de départ.

Le problème SAT paramétré par le nombre de variables n'admet pas d'algorithme sous-exponentiel sauf si ETH est fausse. Le théorème 5 permet de conclure. □

4.4 Discussion

Pour améliorer les résultats présentés dans ce chapitre, il serait intéressant de trouver un algorithme FPT plus rapide pour Common. Il serait également intéressant d'étendre les résultats concernant la kernelization à la notion de Turing kernelization présentée dans la discussion du chapitre 3.

On pourrait également essayer de trouver des paramètres plus intéressants et descendre dans la hiérarchie de paramètres pour trouver d'autres algorithmes FPT pour Common et inversement remonter dans la hiérarchie de paramètre pour espérer trouver des paramètres tels que les problèmes Cons-Among, Cons-Roots et Cons-Common possèdent des noyaux.

Pour finir ce mémoire, nous aimerons abordé un problème ouvert intéressant. Nous nous sommes penchés sur ce problème suite à une discussion concernant un paramètre que nous avions abordé dans l'étude bibliographique. Ce paramètre en question s'appelle « nombre de trous », il fut introduit dans l'article [3].

Le théorème 4 de l'article [3] implique que le problème Cons-Ψ paramétré par le nombre de trous est FPT pour de nombreuses contraintes globales Ψ possédant une certaine propriété. Gaspers et Szeider on prouvé que les problèmes Cons-AtMostNValue

et Cons-NValue paramétrés par le nombre de trous admettent un noyau polynomial [20].

Rappelons brièvement la définition de ce paramètre : soit D un ensemble totalement ordonné, un trou dans un sous-ensemble $D' \subseteq D$ est un couple $(v, w) \in D' \times D'$ tel que $]v, w[\cap D \neq \varnothing$ et $]v, w[\cap D' = \varnothing$.

Notons $\mathit{trous}(S)$ le nombre de trous de S et définissons le paramètre **trous** $= \sum_{x \in \mathcal{V}} \mathit{trous}(\mathcal{D}(x))$.

Le paramètre **trous** mesure la distance entre domaines et intervalles, cependant il dépend fortement de la relation d'ordre totale sur $\mathcal{D}$ utilisée.

Nous avons donc eu l'idée d'introduire un nouveau paramètre, mesurant également la distance entre intervalles et domaines ne dépendant pas de la relation d'ordre utilisée.

Ce paramètre est le nombre minimum de valeurs à supprimer de l'union des domaines des variables afin que chaque variable puisse avoir comme domaine un intervalle. Plus précisément, une fois ces valeurs supprimées, on peut ordonner les valeurs restantes telles que chaque variable ait un intervalle comme domaine.

Calculer ce paramètre est un problème $\mathcal{NP}$-complet cependant nous pensons qu'il est FPT paramétré par la taille de la solution, ce problème peut être vu comme un problème graphique ou encore comme un problème sur les matrices binaires dont les définitions sont les suivantes :

- Convex Bipartite Deletion : Soit $G = (X \cup Y, E)$ un graphe biparti et k un entier. Existe-t'il un sous-ensemble $S \subseteq Y$ tel que $|S| \leq k$ et tel que $G \backslash S$ soit un graphe biparti convexe ? Le graphe biparti $G \backslash S$ est convexe ssi les sommets de $Y \backslash S$ peuvent être ordonné tels que le voisinage de chaque sommet de X est consécutif.

- Min-COS-C : Soit M une matrice binaire. Existe-t'il un ensemble de k colonnes de M tel que la matrice M' obtenu en supprimant cet ensemble de colonnes de M ait la propriété *consecutives ones property* ?

  Une matrice possède cette propriété ssi il existe une permutation de ces colonnes telle que sur chaque lignes les uns sont consécutifs.

Savoir si ces deux problèmes paramétrés par k sont FPT est une question ouverte.

Ces deux problèmes sont très proches du problème Interval Deletion qui consiste à supprimer k sommets d'un graphe pour obtenir un graphe d'intervalle. Interval Deletion paramétré par k est FPT [10]. Nous avons essayer d'utiliser la même approche sur le problème Convex Bipartite Deletion qui utilise les décompositions modulaires.

Remarquons que le problème Min-COS-R paramétré par k qui consiste à supprimer k lignes d'une matrice pour obtenir une matrice avec la propriété *consecutives ones property* est FPT [30]. Ce résultat permet de trouver une paramétrisation intéressante du problème Cons-AtMostNValue tel que ce problème soit FPT mais faute de temps et d'espace, ce résultat ne figure pas dans ce mémoire.

Bibliographie

[1] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Among, common and disjoint constraints. In *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP, 2005, Uppsala, Sweden, June 20-22, 2005, Revised Selected and Invited Papers*, pages 29–43, 2005.

[2] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. The ROOTS constraint. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, pages 75–90, 2006.

[3] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. The parameterized complexity of global constraints. *CoRR*, abs/0903.0467, 2009.

[4] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of global constraints. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 112–117, 2004.

[5] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2) :239–259, 2007.

[6] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8) :423–434, 2009.

[7] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1) :277–305, 2014.

[8] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35) :4570–4578, 2011.

[9] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems : Algorithms and applications. *European Journal of Operational Research*, 119(3) :557–581, 1999.

[10] Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *CoRR*, abs/1211.5933, 2012.

[11] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8) :1346–1367, 2006.

[12] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover : Further observations and further improvements. In *Graph-Theoretic Concepts in Computer Science, 25th International Workshop, WG '99, Ascona, Switzerland, June 17-19, 1999, Proceedings*, pages 313–324, 1999.

[13] Stephen Cook. The p versus np problem. *The millennium prize problems*, page 86, 2006.

[14] Brouno Courcelle. Handbook of theoretical computer science (vol. b). chapter Graph Rewriting : An Algebraic and Logic Approach, pages 193–242. MIT Press, Cambridge, MA, USA, 1990.

[15] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[16] Michael R. Fellows. Parameterized complexity : The main ideas and some research frontiers. In *Algorithms and Computation, 12th International Symposium, ISAAC, 2001, Christchurch, New Zealand, December 19-21, 2001, Proceedings*, pages 291–307, 2001.

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[18] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct {PCPs} for {NP}. *Journal of Computer and System Sciences*, 77(1) :91 – 106, 2011. Celebrating Karp's Kyoto Prize.

[19] Serge Gaspers, Sebastian Ordyniak, M. S. Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-horn. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 67–79, 2013.

[20] Serge Gaspers and Stefan Szeider. Guarantees and limits of preprocessing in constraint satisfaction and reasoning. *CoRR*, abs/1406.3124, 2014.

[21] Georg Gottlob, Francesco Scarcello, and Martha Sideri. Fixed-parameter complexity in ai and nonmonotonic reasoning. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR '99, pages 1–18, London, UK, UK, 1999. Springer-Verlag.

[22] Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *The Computer Journal*, 51(3) :303–325, 2008.

[23] Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. Hierarchies of inefficient kernelizability. *CoRR*, abs/1110.0976, 2011.

[24] Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (turing) kernelization. In *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, pages 202–215, 2013.

[25] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2) :367–375, 2001.

[26] Bart Maarten Paul Jansen. *The Power of Data Reduction : Kernels for Fundamental Graph Problems*. PhD thesis, Universiteit Utrecht, 2013.

[27] Michael Lampis and Valia Mitsou. The computational complexity of the game of set and its theoretical applications. In *LATIN 2014 : Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, pages 24–34, 2014.

[28] Richard J Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM journal on computing*, 9(3) :615–627, 1980.

[29] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105 :41–72, 2011.

[30] N. S. Narayanaswamy and R. Subashini. Obtaining matrices with the consecutive ones property by row deletions. *Algorithmica*, 71(3) :758–773, 2015.

[31] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[32] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 362–367, 1994.

[33] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2) :103–114, 2010.

[34] Mathias Weller. *Aspects of Preprocessing Applied to Combinatorial Graph Problems*. PhD thesis, 2013.