Stage de Recherche en Informatique

Fixed-parameter tractability of counting small minimum (S,T)-cuts

Benjamin Mouscadet

LRI, CentraleSupélec, Université Paris-Saclay

Contexte du stage

Il s'agissait d'un stage de deuxième année au Laboratoire de Recherche en Informatique, sous la direction du Pr. Joanna Tomasik. Si la plupart des stages se font en entreprise, j'ai préféré effectuer un stage de recherche, et travaillé pendant 2 mois avec Pierre Bergé, doctorant, sur un des problèmes qui constitueront sa thèse. J'ai également participé à la rédaction d'une publication scientifique, issue du problème abordé.

Pourquoi faire un stage de recherche ?

1. La recherche est un monde complètement différent, avec lequel les étudiants ne sont que peu familiarisés
2. Les opportunités de faire de la recherche en école d'ingénieur ne sont pas si nombreuses, mais permettent de se faire une idée plus précise de ce qui est attendu d'un chercheur, ou d'un doctorant
3. La recherche confronte chacun avec ses propres capacités, les seules limites sont celles de l'imagination et la rigueur du chercheur. C'est un travail en autonomie
4. La recherche est un monde intellectuellement très stimulant, et donne la possibilité de résoudre des problèmes sans limite de difficulté
5. Un stage de recherche permet de comprendre mieux le système des publications auquel sont soumis tous les chercheurs
6. Les doctorants sont une population jeune et dynamique avec qui échanger est très stimulant, chacun offrant un point de vue unique sur son domaine d'expertise

Position du problème

Donné un graphe $G = (V, E)$ et deux ensembles $S, T \subseteq V$ de noeuds du graphe, on définit :

- une *cut* $X \subseteq E$ est un ensemble d'arêtes du graphe telle qu'il n'existe aucun chemin reliant S à T dans le graphe G privé de X
- une *mincut* est une cut (non nécessairement unique) dont le nombre d'arêtes est minimal

Dans ce problème, on cherche un algorithme qui **compte toutes les mincuts** du graphe.

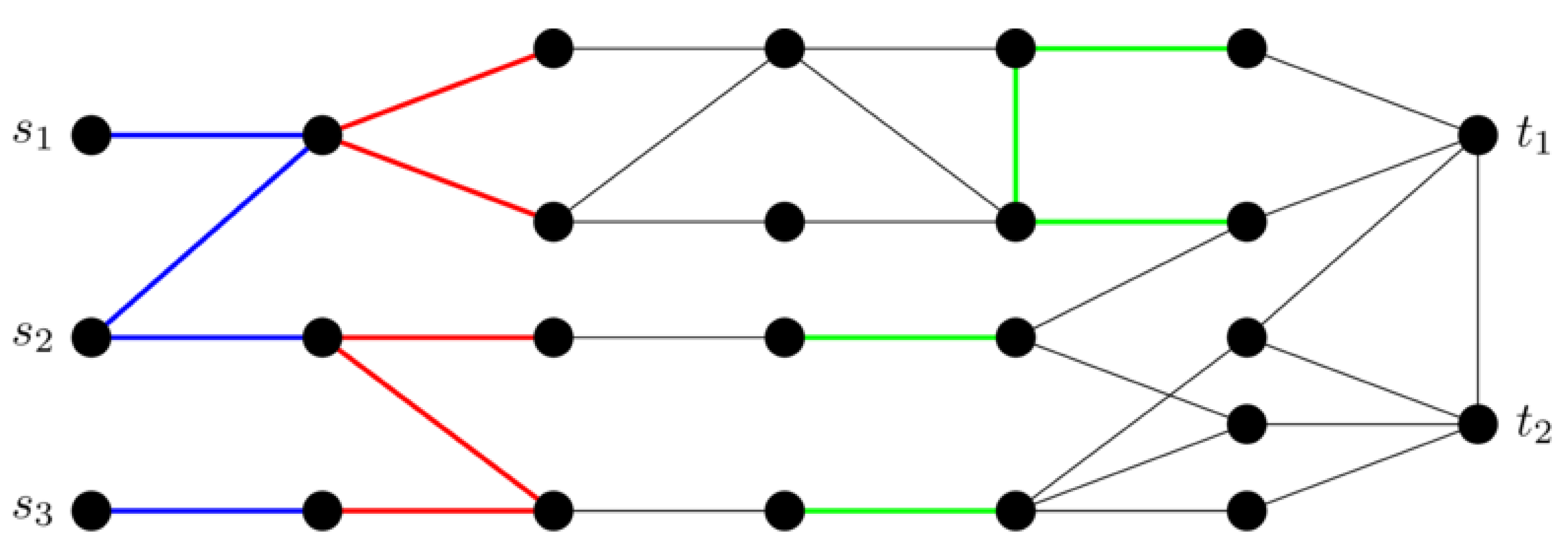

Figure 1: Deux exemples de cuts (en rouge et vert, taille 5) et une mincut (en bleu, taille 4)

Complexité

- Trouver la taille des *mincuts* est un problème **P** : cela peut être fait en temps polynomial n^3 où $n = |V|$ est le nombre d'arêtes du graphe
- En revanche compter le nombre total de *mincuts* est un problème **NP-HARD** : il n'existe pas d'algorithme polynomial pour le faire

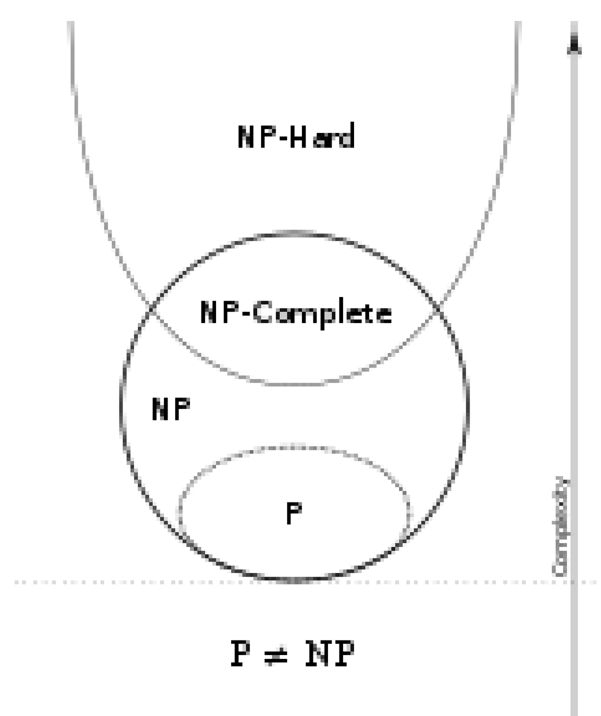

Figure 2: Les différentes classes de complexité

Alors que la solution exacte d'un problème **P** peut être obtenue en temps raisonnable sur des instances de grande taille ($n = 10^4$ par exemple), la résolution d'un problème **NP-HARD** nécessitera généralement un temps exponentiel en la taille de l'instance.

Complexité	Dénomination	$n = 5$	$n = 10$	$n = 50$	$n = 1000$	$n = 10000$	$n = 1000000$
$O(log(n))$	Logarithmique	10ns	10ns	20ns	30ns	40ns	60ns
$O(n)$	Linéaire	50ns	100ns	500ns	$10mus$	$100\mu s$	$10ms$
$O(n^2)$	Quadratique	$250ns$	$1\mu s$	$25\mu s$	$10ms$	$1s$	$2,8h$
$O(n^3)$	Cubique	$1,25\mu s$	$10\mu s$	$1,25ms$	$10s$	$2,7h$	316 ans
$O(2^n)$	Exponentielle	$320ns$	$10\mu s$	130 jours	10^{59}ans	—	—
$O(n!)$	Factorielle	$1,2\mu s$	$36ms$	10^{48}ans	—	—	—

Table 1: Temps de calcul pour différentes complexité, à 10^8 flops (opérations/seconde). En rouge les complexités polynomiales ou sub-polynomiales

Approches possibles

On a alors deux possibilités :

1. Utiliser des **algorithmes d'approximations** : donnent une solution approchée en temps polynomial
2. Utiliser des **algorithmes exacts mais polynomiaux pour certaines instances du problème seulement**

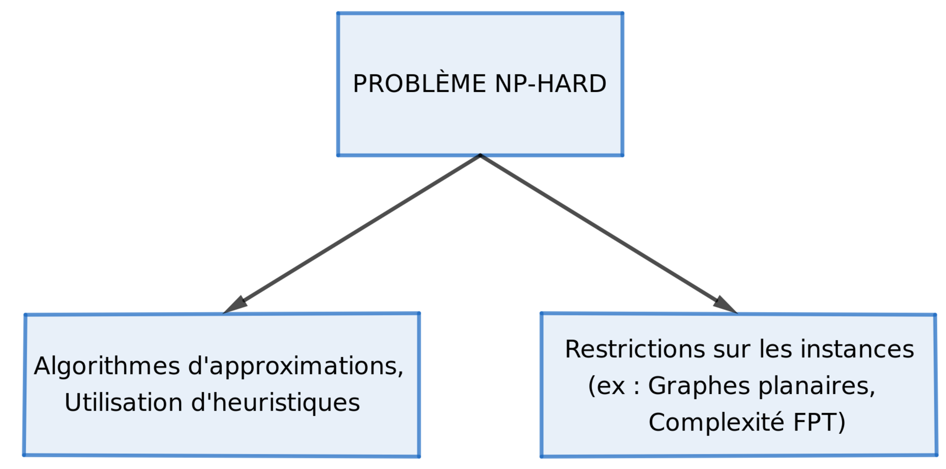

Figure 3: Plusieurs solutions possibles lorsqu'on est confrontés à un problème NP-HARD

- On choisit de travailler sur les instances dont la taille p de la mincut est petite : on s'autorise donc à faire intervenir des termes exponentiels en p dans la complexité
- En plus de la taille n de l'instance, on ajoute p comme paramètre de complexité. Pour $p << n$, les temps de calculs sont raisonnables. On parle de **Complexité FPT** lorsqu'elle s'ecrit sous la forme $O\left(Q(n)f(p)\right)$, où Q est un polynôme et f une fonction arbitraire.

Taille paramètres	$n = 100$	$n = 1000$	$n = 10000$
$p = 1$	10 ms	10 s	$2,8$ h
$p = 10$	10 s	$2,8$ h	115 jours
$p = 100$	10^{20} ans	10^{23} ans	10^{26} ans

Table 2: Quelques temps de calcul pour un algorithme FPT en $O(2^p n^3)$, à 10^8 flops

Algorithme et méthodes

1. On commence par opérer une réduction pour obtenir un graphe "à étages"
   - On trouve une *mincut* X_i la plus proche de S possible, et tous les noeuds entre S et les extrémités gauche de X_i sont regroupés dans un "étage" R_i
   - Les extrémités droite de X_i deviennent le nouvel ensemble S. On itère jusqu'à ne plus pouvoir trouver de *mincut*

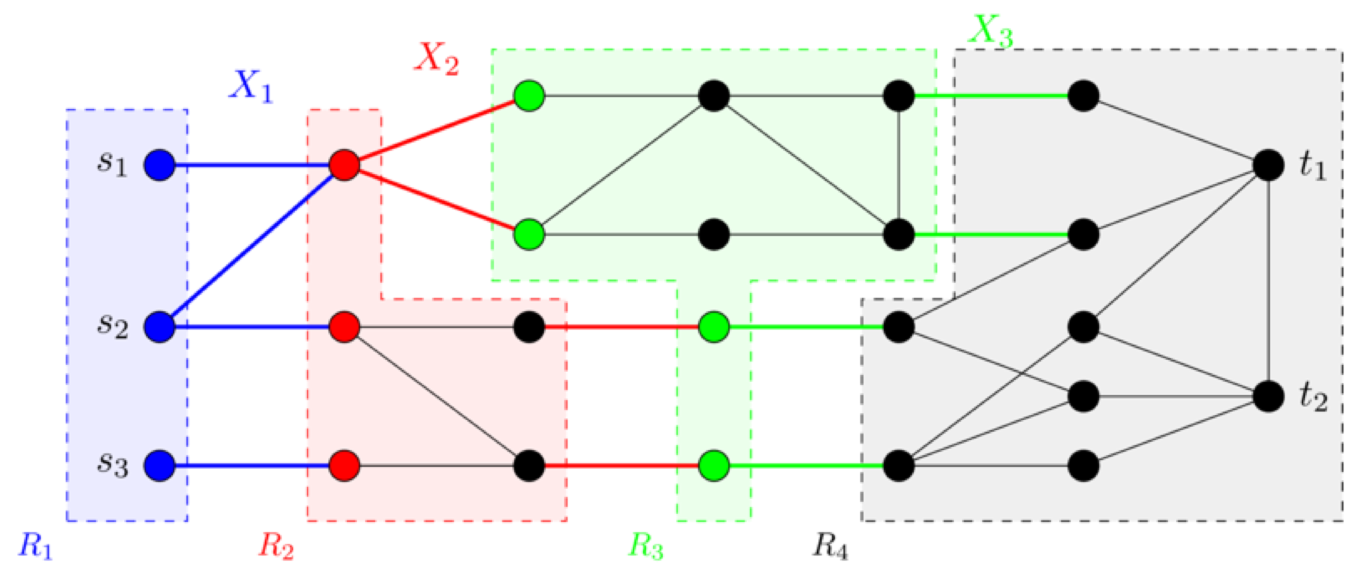

Figure 4: Décomposition en graphe à étages

2. On oriente partiellement le graphe en calculant un ensemble maximum de chemins edge-disjoints, qui ont une propriété fondamentale : **le cardinal de cet ensemble maximum est égal à la taille de la mincut, et chaque chemin passe une et une seule fois par chaque mincut**

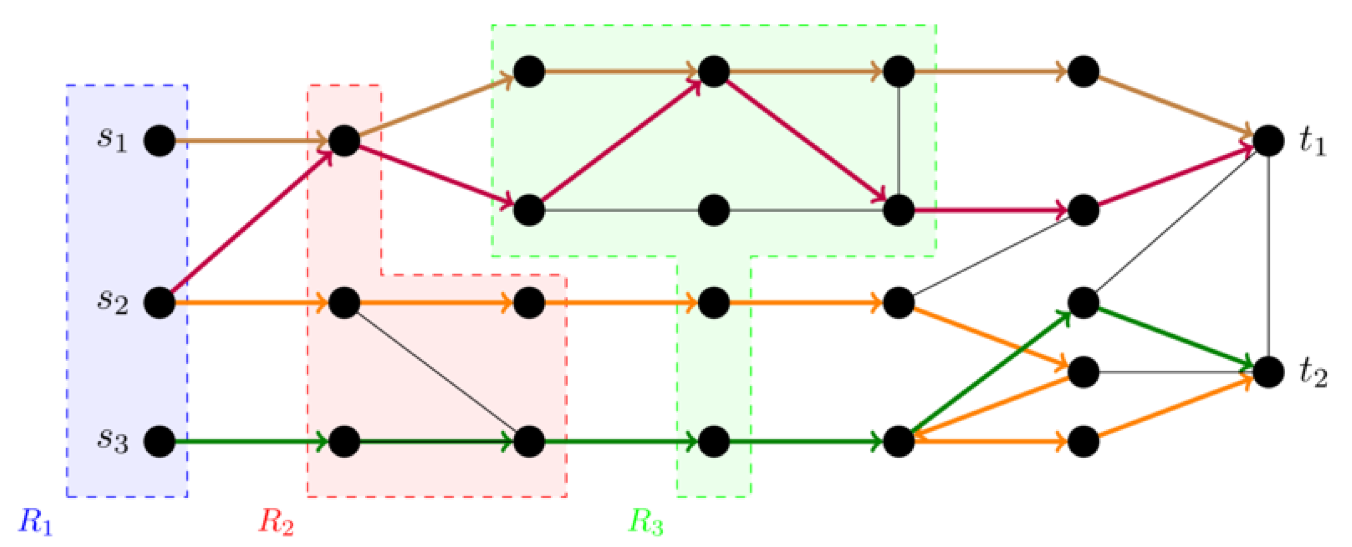

Figure 5: Orientation du graphe selon les 4 chemins edge-disjoints

3. Chaque *mincut* X peut alors être partitionnée en $X = B \cup C$ de sorte que
   - Il existe i tel que $B \subseteq X_i$, et B contient les arêtes de X les plus à gauche
   - Si on supprimait $X_i \backslash B$ du graphe, toutes les arêtes de C seraient inaccessibles depuis S dans le graphe orienté
4. Avec cette propriété et de la combinatoire, on peut compter toutes les *mincut*

À titre indicatif, **on obtient une complexité en** $2^{O(p^2)} n^{O(1)}$; c'est à dire qu'à p constant, la complexité est un polynôme de n multipliée par une constante exponentielle en p^2.

Perspectives

- Bien que la complexité obtenue reste grande, elle prouve que le problème est FPT, et ouvre la voie à des recherches en vue de trouver un algorithme plus optimisé
- Au delà de l'aspect purement intellectuel, la résolution de ce problème a des implications dans le domaine du traitement de l'image notamment