Rapport de stage : Arbres de décision quantiques

Titouan Carette

Encadrant : Frédéric Magniez

2 septembre 2015

Introduction

Une fois l'idée d'utiliser les phénomènes quantiques pour calculer émise, la question la plus immédiate est « Un tel ordinateur serait-il plus puissant qu'un ordinateur classique ? ». L'algorithme de Shor et son gain exponentiel en complexité par rapport aux meilleurs algorithmes classiques, puis l'algorithme de Grover et son gain quadratique donnèrent un début de réponse à cette question. Mais encore faut-il savoir jusqu'où exactement peut aller un ordinateur quantique.

C'est là qu'interviennent les modèles de complexité, et parmi eux celui de la complexité en requête, qui s'intéresse au nombre d'interrogations de l'entrée effectuées par un algorithme lors de son exécution. Pour trouver une borne supérieure sur la complexité d'un problème, il suffit d'exhiber un algorithme. Cependant, trouver une borne inférieure est bien plus délicat. Le formalisme mathématique de la physique quantique nous fournit pourtant une méthode générale pour trouver des bornes sur les complexités : l'adversaire quantique.

Mais cette méthode est peu intuitive et très rapidement complexe pour les problèmes non triviaux. Ainsi nous connaissons exactement la complexité de la recherche d'éléments dans une liste, mais celle de la recherche de triangle dans un graphe, qui nous intéresse ici, est toujours une question ouverte. La meilleure borne inférieure connue est $\Omega(n)$ et toute la difficulté de trouver une borne supérieure est de concevoir un algorithme quantique efficace. Or, les phénomènes quantiques étant peu intuitifs, la conception de tels algorithmes est délicate. Il est alors utile d'avoir à disposition des modèles intermédiaires dans lesquels l'intuition classique permet d'obtenir des algorithmes quantiques. C'est le cas, par exemple, des marches aléatoires quantiques et du modèle qui nous intéresse ici : les learning graphs. Introduits par Belovs [5], ces dérivés de la méthode de l'adversaire peuvent être vus comme des arbres de décision quantiques qui transforment la conception d'un algorithme en problème combinatoire.

Plus de la moitié de ce stage a consisté en un travail bibliographique afin de se familiariser avec les différentes notions d'algorithmique quantique en général, et le modèle des learning graphs en particulier. La première partie de ce rapport introduit les prérequis nécessaires à la manipulation des learning graphs. La seconde partie expose ce qui a été concrètement effectué pendant le stage, à savoir la retranscription de l'algorithme de Le Gall sous la forme d'un learning graph adaptatif et la mise en forme de différents lemmes qui simplifient le calcul de sa complexité.

Je profite de la fin de cette introduction pour remercier Frédéric Magniez pour ses patientes et nombreuses explications, ses conseils et surtout pour m'avoir offert l'occasion de m'immiscer dans un domaine en pleine effervescence. Je remercie aussi Mathieu Laurière pour son accueil chaleureux et les discussions éclairantes, ainsi que Noëlle Delgado pour son aide au cours des diverses formalités administratives.

Table des matières

1 Intoduction aux learning graphs

1.1 Complexité en requête

1.1.1 Cas classique

Lors de l'analyse de la complexité d'un algorithme, il est essentiel d'énoncer clairement ce que l'on considère comme coûteux en ressource ou au contraire négligeable. Dans le modèle de la complexité en requête, on considère qu'une fois une entrée donnée, les seules opérations coûteuses sont les accès à cette entrée, les requêtes. Le coût de toute autre manipulation des données n'est pas pris en compte.

Définition 1 *La complexité en requête d'un problème est le nombre minimal de requêtes nécessaires à un algorithme pour résoudre le problème dans le pire cas.*

Prenons l'exemple très simple du tri d'une liste d'entiers. L'entrée est ici une liste de n entiers, et une requête consiste en la comparaison de deux éléments. Nous savons alors que la complexité en requête est en $O(n\log(n))$. Cela ne signifie pas qu'on ne peut concevoir d'algorithmes plus performants, mais seulement que si de tels algorithmes existent, alors ils font appel à des idées qui ne sont pas purement liées aux requêtes. Ainsi, il existe des algorithmes qui trient dans certains cas plus rapidement mais en ayant recours à d'autres aspects que la seule comparaison entre les éléments.

1.1.2 Cas quantique

En mécanique quantique, un état quantique ϕ est représenté par un vecteur dans un espace hermitien noté $|\phi\rangle$. Nous pouvons agir sur celui-ci soit par une transformation unitaire (on multiplie le vecteur par une matrice unitaire), soit par mesure selon une base ; ainsi, imaginons que notre état ϕ évolue dans un espace de dimension complexe n et donnons-nous une base orthonormée $|\psi_i\rangle$ telle que $|\phi\rangle = \sum_{i=1}^{n} a_i|\psi_i\rangle$: mesurer selon la base $|\psi_i\rangle$ signifie observer l'état ϕ qui apparaîtra alors comme $|\psi_i\rangle$ avec probabilité $|a_i|^2$.

On peut donc représenter un algorithme quantique comme une suite de transformations unitaires appliquées à un état de départ fixé, suivie d'une mesure finale selon une base donnée. On définit donc une transformation unitaire *oracle* qui permet d'accéder à l'entrée.

Définition 2 *La complexité en requête quantique est le nombre minimal de recours à l'oracle nécessaires dans le pire cas pour obtenir un état qui, une fois mesuré, nous donnera la bonne réponse avec une probabilité strictement supérieure à une constante fixée c, avec $c > \frac{1}{2}$.*

On a bien ici cette même idée de compter le nombre d'accès à l'entrée.

1.2 Certificat

Nous abordons maintenant la complexité en requête de manière plus formelle. Nous ne nous intéresserons, tout au long de ce rapport, qu'à des problèmes de décision que l'on peut identifier à des fonctions booléennes

renvoyant 1 si la réponse est « oui » et 0 si « non ».
Un algorithme est une méthode pour calculer cette fonction en prenant son argument en entrée. On considère les entrées comme des n-uplets de booléens. Une requête est un accès de l'algorithme à l'une des composantes.

Considérons l'exemple de la recherche d'un élément dans une liste de taille n. L'entrée est alors un n-uplet dont la i-ème composante vaut 1 si le i-ème élément est celui recherché et 0 sinon. Une requête revient ici à vérifier si un élément est bien celui recherché et l'algorithme renvoie 1 si l'élément est trouvé et 0 sinon.

On comprend bien qu'il n'est pas toujours nécessaire d'interroger l'intégralité de l'entrée pour connaître de façon certaine la valeur de la fonction. Il suffit souvent de connaître certaines composantes bien particulières. On appelle un tel ensemble de composantes un 1-certificat. Plus formellement :

Définition 3 *Soient une fonction f et une entrée x. Un ensemble c de composantes est un 1-certificat pour x si pour toute entrée y telle que les composantes dans c de y soient égales à celles de x, on a $f(y) = 1$.*

Bien sûr, une entrée z telle que $f(z) = 0$ n'admet pas de 1-certificat. On remarque que les 1-certificats apportent beaucoup d'information sur la fonction et les éventuels algorithmes qui pourraient la calculer. Pour capturer cette idée, on définit la structure de certificat d'une fonction.

Définition 4 *Un ensemble C est la structure de certificat d'une fonction f si pour toute entrée $x \in f^{-1}(\{1\})$ il existe $M \in C$ tel que tout élément de M soit un 1-certificat pour x, on dit que les éléments de M sont marqués.*

En fait, il s'agit de la collection de tous les ensembles de requêtes pouvant amener à conclure que f vaut 1 sur une entrée. Plusieurs fonctions peuvent avoir la même structure de certificat. On définit alors la complexité en requête d'une structure de certificat comme le maximum des complexités en requête des fonctions ayant cette structure de certificat. Dès lors, un algorithme ne prenant en compte que la structure de certificat ne pourra faire mieux que la complexité en requête de la structure. Pour être plus performant, d'autres aspects inhérents à la fonction devront être pris en compte.

En conservant l'exemple de la recherche d'un élément dans une liste de taille n, on remarque que toute entrée de $f^{-1}(\{1\})$ admet un singleton comme 1-certificat ; en effet, si une requête est positive, c'est terminé. Tout ensemble contenant ce singleton est alors un 1-certificat pour cette entrée. Ainsi la structure de certificat pour cette fonction se présente comme un ensemble C contenant n ensembles M_i qui contiennent eux-mêmes toutes les parties du segment $[\![1, n]\!]$ contenant i. On appelle cette structure $OR - certificate\ structure$ car il s'agit de celle de la fonction logique « ou ».

1.3 Learning graphs

Un learning graph peut être vu comme un arbre de décision quantique, une façon de comprendre le fonctionnement d'un algorithme quantique en termes de requête. Le principal intérêt est qu'à un learning graph correspond une marche quantique, un algorithme concret permettant de résoudre le problème. Le learning graph est donc un moyen de transformer le problème de la conception d'un algorithme quantique en un problème combinatoire plus intuitif.

1.3.1 Définition

On notera un learning graph $G_L = (V_L, E_L)$ afin d'éviter les confusions lorsque d'autres graphes entreront en jeu. Un learning graph dépend de la taille de l'entrée, on considère donc des entrées de taille n représentées par des n-uplets. Si on représente l'état de l'algorithme par l'ensemble des requêtes déjà effectuées, alors $\mathcal{P}([\![1, n]\!])$ est l'ensemble des différents états possibles de l'algorithme.

Un learning graph est un graphe pondéré aux arêtes dont les sommets sont identifiés aux états de l'algorithme. On identifiera par exemple $\emptyset$, l'état initial de l'algorithme, et le sommet qui le représente. Deux états sont reliés, si est seulement si, ils ne diffèrent que d'une requête ; autrement dit, si l'un des deux ensembles contient exactement un élément de plus que l'autre. Formellement :

Définition 5 *Un learning graph est le diagramme de Hasse représentant $\mathcal{P}([\![1,n]\!])$ muni de la relation d'inclusion stricte, que l'on pondère par une fonction $w : E_L \to \mathbb{R}_+$. Lorsque la fonction w dépend de l'entrée x, le learning graph est dit adaptatif, sinon il est dit non-adaptatif.*

Cette distinction aura son importance par la suite.

1.3.2 Flot

On définit, pour chaque entrée $x \in f^{-1}(\{1\})$, un flot réel positif q_x sur le learning graph. Pour se faire le learning graph est considéré comme un graphe orienté.

Définition 6 *La fonction $q_x : V_L \times V_L \to \mathbb{R}_+$ est un flot pour l'entrée x sur le learning graph $G_L = (V_L, E_L)$ si elle vérifie :*

1) $\forall u \in V_L, \forall v \in V_L, q_x(uv) = -q_x(vu)$.

2) Le flot est unitaire et $\emptyset$ est l'unique source du flot, ainsi : $\sum_{v \in N(\emptyset)} q_x(\emptyset v) = 1$.

3) Le flot se conserve en chaque sommet sauf en $\emptyset$, la source, et dans les sommets de M, qui sont les puits du flot. Il en suit que pour tout autre sommet v : $\sum_{u \in N(v)^-} q_x(uv) = \sum_{w \in N(v)^+} q_x(vw)$.

On peut voir le flot comme une représentation du fonctionnement de l'algorithme. Tout l'intérêt de la dimension quantique de l'algorithme est de permettre d'effectuer simultanément plusieurs requêtes. Le flot montre les requêtes utiles, celles qui aboutissent à la découverte d'un 1-certificat. On noteras par la suite par la suite q pour parler de tout les flots q_x.

1.3.3 Complexité

L'intérêt des définitions précédentes est d'introduire la complexité d'un learning graph. Il est en fait possible de concevoir un algorithme quantique résolvant le problème associé et ayant pour complexité en requête un grand O de cette complexité. Grâce à ce modèle, concevoir un algorithme quantique efficace revient alors à choisir astucieusement les poids et le flot afin de minimiser la complexité du learning graph.

Complexité négative

On définit la complexité négative C_0 d'un learning graph comme la somme des poids de ses arêtes. Afin de rester le plus général possible, nous considérerons le cas adaptatif : les poids w dépendent donc de x et on notera w_x les poids. La complexité négative sera alors notée $C_0(x)$.

Définition 7 *La complexité négative d'un learning graph est $C_0(x) = \sum_{e \in E_L} w_x(e)$.*

Dans le cas non adaptatif la complexité négative d'un learning graph a un sens, et ne dépend pas de x, mais dans le cas adaptatif, elle n'est qu'un intermédiaire dans le calcul de la complexité totale, et elle dépend de l'entrée x.

Complexité positive

Si la complexité négative ne prend en compte que les poids, la complexité positive prend en compte le flot. Soit un flot q_x définit pour une entrée $x \in f^{-1}(\{1\})$ sur le learning graph. On définit l'énergie $\mathcal{E}(f_x)$ du flot.

Définition 8 *L'énergie d'un flot q_x sur un learning graph est $\mathcal{E}(q_x) = \sum_{e \in E_L} \frac{q_x(e)^2}{w_x(e)}$.*

Ici le learning graph est considéré comme non-orienté et chaque arête est comptée une seule fois. La complexité positive du learning graph pour l'entrée x, $C_1(x)$, est définie comme l'énergie minimale d'un flot pour x.

Définition 9 *La complexité positive d'un learning graph est $C_1(x) = \min_{q_x} \mathcal{E}(q_x)$.*

De même que pour la complexité négative, dans le cas non adaptatif, cela a du sens de parler de la complexité positive d'un learning graph. Elle est alors définie comme le maximum sur les entrées des complexités positives.

$$C_1 = \max_{x \in f(1)^{-1}} C_1(x) \tag{1.1}$$

Dans le cas adaptatif, il s'agit aussi d'un intermédiaire dans le calcul de la complexité totale.

Complexité totale

Une fois introduites les complexités négatives et positives, la complexité totale peut alors être définie comme le maximum ou la moyenne géométrique de ces deux complexités. Ces deux définitions sont en fait équivalentes. Dans le cas non-adaptatif, où les complexités négative et positive d'un learning graph sont bien définies, il suffit de prendre directement la moyenne géométrique.

Définition 10 *La complexité d'un learning graph non-adaptatif est $C = \sqrt{C_0 C_1}$.*

Cependant dans le cas adaptatif il faut procéder comme suit :

Définition 11 *La complexité d'un learning graph adaptatif est $C = \sqrt{\left(\max_{x \in f^{-1}(\{0\})} C_0(x)\right)\left(\max_{x \in f^{-1}(\{1\})} C_1(x)\right)}$.*

Il est intéressant de constater ce qui se passe lorsque l'on multiplie tous les poids par un même nombre non nul α. Cela revient a multiplier par α la complexité négative et diviser par α la complexité positive ; il n'y a donc aucun impact sur la complexité totale. Nous utiliserons cette opération à de nombreuses reprises par la suite.

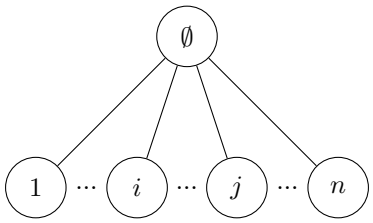
Il a été dit que nous pourrons ensuite transformer un learning graph de complexité $O(C)$ en un algorithme quantique de même complexité. On définit alors la complexité en terme de learning graph d'une structure de certificat comme la complexité minimale d'un learning graph pour cette structure.

1.4 Utilisation pratique des learning graphs

En pratique, de nombreux aménagements du modèle sont faits pour simplifier l'analyse. Un learning graph, sur une entrée de taille n, devrait avoir 2^n sommets. Cependant parmi ceux-ci, certains ne sont jamais atteints par aucun flot, de même que certaines arêtes ne sont jamais traversées. On peut alors considérer que le poids de ces arêtes est nul. Ne convoyant aucun flot, elles n'ont aucune incidence sur la complexité du learning graph et on choisit de ne pas les représenter. On ne représente pas non plus les sommets dont toutes les arêtes incidentes sont de poids nul.

1.4.1 Recherche de Grover

La recherche d'un élément dans une liste de taille n et un des problèmes les plus classique de l'algorithmique quantique. L'algorithme de Grover permet de le résoudre en $O(\sqrt{n})$ et il est optimal, les learning graphs permettent d'obtenir le même résultat. Voici comment on représente le learning graph correspondant au problème de recherche d'un élément dans une liste de taille n.



Seules les parties utiles sont représentées.

Lemme 1 *La complexité en terme de learning graph de la recherche de Grover sur une lste de taille n est en* $O(\sqrt{(n)})$.

Preuve. Calculons la complexité, pour des raisons de symétrie, on assigne le même poids à toutes les arêtes. Comme multiplier tout les poids par une constante non nulle ne change pas la complexité, on fixe arbitrairement les poids à 1. Ensuite il suffit de construire un flot pour tout $x \in f^{-1}(\{1\})$. On choisit de ne considérer qu'un unique 1-certificat pour chaque x.

On route naturellement le flot de $\emptyset$ au 1-certificat correspondant à la position de l'élément recherché. Ce chemin ne comportant qu'une arête, on a :

$$C_0 = \sum_{i=1}^{n} 1 = n \tag{1.2}$$

$$C_1 = 1 \tag{1.3}$$

Et finalement :

$$C = \sqrt{n}. \tag{1.4}$$

□

On peut donc à partir de ce learning graph concevoir un algorithme quantique avec une complexité en requête en $O(\sqrt{n})$. On trouve bien le même résultat que l'algorithme de Grover.

Nous nous sommes contentés d'effectuer le calcul pour un x et un flot donnés sans nous préoccuper de l'optimalité. Cette démarche est valide étant donné que c'est une borne supérieure sur la complexité qui nous intéresse. Ici le learning graph est non-adaptatif ; on peut se demander si un learning graph adaptatif pourrait faire mieux.

Il serait tentant de mettre systématiquement un poids nul sur les arêtes ne routant aucun flot. Pourtant, cette approche n'est pas viable. Le poids d'une arête peut en effet dépendre de x, cependant il ne peut dépendre que des requêtes déjà effectuées lorsque l'on traverse cette arête. Concevoir un learning graph adaptatif est donc assez délicat et dans le cas de la recherche d'élément, le learning graph non-adaptatif est bien optimal.

Nous utiliserons très souvent ce type de learning graph par la suite, notamment comme sous-routine au sein de learning graphs plus complexes. Pour simplifier l'écriture, on représentera une recherche de Grover par une flèche simple en cachant l'arborescence des possibilités.

$$\emptyset \xrightarrow{Grover}$$

1.4.2 Calcul pratique de la complexité d'un learning graph

Avant de calculer la complexité de learning graphs plus imposants, nous allons introduire plusieurs notions et méthodes qui simplifient grandement l'analyse et seront utilisées abondamment tout au long de ce rapport.

Sommets redondants

Il est parfois commode pour clarifier la représentation et l'analyse d'un learning graph que plusieurs sommets s_i représentent le même ensemble. Cela n'est pas gênant, on peut en effet se ramener à un learning graph régulier de complexité inférieure par quelques manipulations simples.

Lemme 2 *Pour tout learning graph comportant des sommets redondants ont peut construire un learning graph régulier de complexité inférieure.*

Preuve. S'il existe plusieurs arêtes e_i de poids w_i reliant des sommets représentant un ensemble S et S' alors on construit un learning graph où ces sommets sont reliés par une arête e de poids $\sum_i w_i$. Ainsi la complexité négative des deux graphes est égale. Quant au flot, si ces arêtes sont traversées par un flot $q_x(e_i)$ alors on route à travers la nouvelle arête un flot $\sum_i q_x(e_i)$. Le nouveau flot est bien conforme à la définition et de plus l'inégalité :

$$\frac{\left(\sum_i q_x(e_i)\right)^2}{\sum_i w_i} \leq \sum_i \frac{q_x(e_i)^2}{w_i}$$

nous garantit que la complexité positive et, finalement, la complexité totale du nouveau learning graph seront inférieures à celles de l'ancien. □

On peut donc travailler sur un learning graph avec sommets redondants pour calculer une borne supérieure sur la complexité.

Transition

Souvent, plusieurs requêtes nécessitent d'être faites à la suite de manière systématique, par exemple lorsque l'on interroge directement un ensemble de variables. Sur un learning graph cela apparaît comme un chemin au sens des graphes. Afin de simplifier l'écriture, on a l'habitude de concaténer ces chemins et de les représenter par une arête unique que l'on appellera transition. On définit alors la longueur d'une transition comme le nombre d'arêtes qu'elle représente, et on admet que toute les arêtes au sein d'une transition ont même poids. Ces conventions permettent non seulement de simplifier la représentation d'un learning graph mais aussi son analyse.

Calcul par étage

Afin de calculer la complexité d'un learning graph imposant, il est possible de le découper en graphes plus petits dont on calculera la complexité séparément après avoir fixé un flot sur tout le graphe. La somme

de ces complexités est alors un majorant de la complexité totale.

En effet, soit un learning graph $G_L = (V_L, E_L)$ et des $E_L^{(i)}$, les étages, qui forment une partition de E_L. **Attention** : il est important que le nombre d'étages k ne dépende pas de n, la taille de l'entrée.
On définit des complexités négative et positive par étage, $C_0^{(i)}$ et $C_1^{(i)}$, ainsi qu'une complexité par étage $C^{(i)} = \sqrt{C_0^{(i)} C_1^{(i)}}$.

Lemme 3 *La complexité d'un learning graph partitioné en k étages $E_L^{(i)}$ de complexités respectives $C^{(i)}$, où k est une constante, est en $O\left(\sum_{i=1}^{k} C^{(i)}\right)$.*

Preuve. On commence par diviser dans chaque étage tout les poids par la complexité négative de l'étage. On obtient :

$$C_0 = k \tag{1.5}$$

et

$$C_1 = \sum_{i=1}^{k} C_0^{(i)} C_1^{(i)} \tag{1.6}$$

Or l'inégalité :

$$\sum_{i=1}^{k} C_0^{(i)} C_1^{(i)} \leq \left(\sum_{i=1}^{k} \sqrt{C_0^{(i)} C_1^{(i)}} \right)^2 \tag{1.7}$$

nous donne finalement :

$$C \leq \sqrt{k} \sum_{i=1}^{k} C^{(i)}. \tag{1.8}$$

Donc une complexité en $O\left(\sum_{i=1}^{k} C^{(i)}\right)$. □

Spécialité

S'il est possible de simplifier le calcul de la complexité totale en divisant le graphe en étages, il est aussi possible de simplifier de manière radicale le calcul de la complexité d'un étage sous certaines hypothèses.

Supposons qu'un étage $E_L^{(i)}$ soit un ensemble de transitions parallèles de même longueur L_i et où le flot ne prend que deux valeurs, 0 et une valeur constante c. Par parallèles, on entends que ces transitions forment une coupe du learning graph. De plus M, l'ensemble des 1-certificats, ne doit affecter le flot dans l'étage qu'à une permutation des transitions près. Autrement dit, quelque soit l'entrée, le nombre d'arêtes parcourues par le flot et la valeurs c du flot lui-même doivent demeurer inchangées dans l'étage. On note n_{tot} le nombre total de transitions et n_{utile} le nombre de transition parcourues par un flot non nul.

Définition 12 *La spécialité de l'étage i est définie par : $T_i = \frac{n_{\text{tot}}}{n_{\text{utile}}}$.*

Lemme 4 *Si un étage vérifie les hypothèses précédentes alors sa complexité est en $O\left(L_i \sqrt{T_i}\right)$.*

Preuve. Pour le voir il faut assigner à toutes les arêtes d'un étage un même poids w_i. On a alors :

$$C_0^{(i)} = n_{\text{tot}} L_i w_i \tag{1.9}$$

et

$$C_1^{(i)} = n_{\text{utile}} \frac{c^2}{w_i} \tag{1.10}$$

finalement on a $C^{(i)} = n_{\text{utile}} c L_i \sqrt{T_i}$, or par définition du flot $n_{\text{utile}} c \leq 1$, on obtient $C^{(i)} = O\left(L_i \sqrt{T_i}\right)$. □

Les hypothèses de l'utilisation de la spécialité paraissent très restrictives mais sont pourtant très fréquemment vérifiées et ces méthodes permettent un gain de temps précieux.

1.4.3 Marche sur le graphe de Johnson, l'exemple d'*element distinctness*

Les learning graphs sont étroitement liés aux marches quantiques, les homologues des marches aléatoires classiques. Par exemple, l'algorithme de recherche de Grover peut être vu comme une marche quantique et les algorithmes obtenus à partir des learning graphs en sont également. Les marches sur le graphe de Johnson sont parmi les plus utilisées et seront très présentes, aux même titre que les recherches de Grover, comme sous-routines dans les learning graphs. Ici les learning graphs seront toujours considérés comme non-adaptatifs, bien que tous les concepts présentés se généralisent aisément au cas adaptatif, ce qui sera fait dans la seconde partie de ce rapport.

Graphe de Johnson

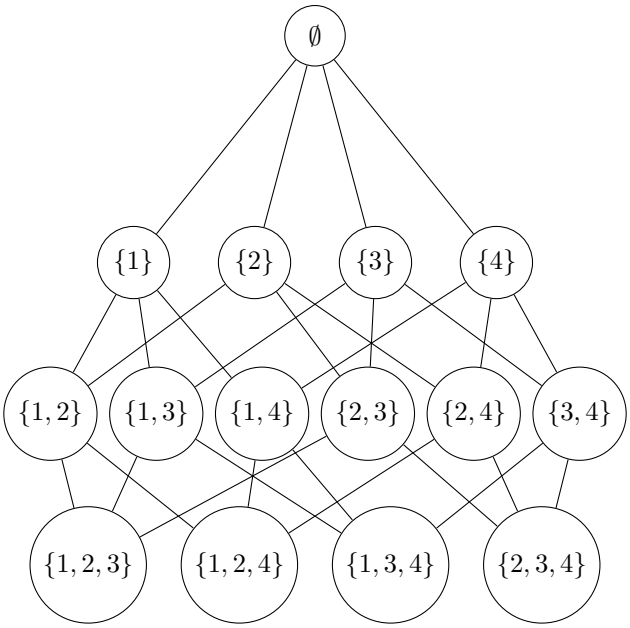
On définit le graphe de Johnson $J(n, r)$ comme le graphe non-orienté dont les sommets correspondent aux ensembles à r éléments dans $\mathcal{P}(\llbracket 1, n \rrbracket)$. Deux ensembles sont reliés si et seulement si ils ne diffèrent que d'un élément, c'est à dire si l'on peut obtenir l'un à partir de l'autre en retirant puis en ajoutant un unique élément. L'idée d'une marche sur le graphe de Johnson est de trouver d'abord un ensemble d'élément auquel on restreindra ensuite les recherches.

Element Distinctness

Nous allons voir comment cette marche se traduit en terme de learning graphs à travers l'exemple d'*element distinctness*. Il s'agit d'un problème simple pour lequel la marche sur le graphe de Johnson est optimale. Étant donnés une fonction h et un ensemble X de taille n, on cherche à savoir s'il existe deux éléments x et y tels que $h(x) = h(y)$. Cela revient par exemple à rechercher une collision pour une fonction de hachage.

Nous choisissons de router le flots vers les 1-certificat correspondant à une unique collision pour chaque $x \in f^{-1}(\{1\})$. Ici les éléments marqués appartenant à M sont tous les ensembles contenant cette paire. Le modèle ne nous obligeant pas à utiliser tout les éléments de M comme puits, nous ne nous intéresserons qu'aux éléments marqués de taille r où r est un $o(n)$.

Le learning graph pour *element distinctness* se présente ainsi pour $n = 4$ et $r = 3$:



Le learning graph se découpe très simplement en trois étages distincts. On commence d'abord par questionner tous les ensembles à $r-2$ éléments. Pour représenter cela de la façon la plus simple possible, on copie les sommets afin d'obtenir de grandes transitions de longueur $r-2$ entre $\emptyset$ et les dits ensembles. Dans une deuxième étape on questionne le $(r-1)-$ème élément et enfin dans une troisième étape, on interroge le dernier élément. Lors de la conception du flot nous ne nous intéresserons qu'aux chemins ne découvrant la collision que lors des étapes deux et trois.

Lemme 5 *La complexité du learning graph correspondant à element distinctness est en* $O(n^{\frac{2}{3}})$.

Preuve. Nous commençons par router le flot de manière uniforme dans les transitions qui n'interrogent aucun des éléments de la collision. Lors de l'étape deux, le flot, toujours uniforme, n'est envoyé qu'à travers les arêtes qui interrogent un élément de la collision. Et enfin le flot traverse l'arête correspondant à la requête sur l'élément de la collision manquant, avant de terminer sa course dans l'un des sommets marqués.

Maintenant que le flot est fixé, nous allons calculer la complexité par étage. Remarquons que ceux-ci vérifient toutes les hypothèses nécessaires à l'utilisation de la spécialité :

Étage 1 : La longueur L_1 est de $r-2$. Il y a $\binom{n}{r-2}$ transitions et $\binom{n-2}{r-2}$ transitions où circule un flot non nul. La spécialité T_1 vaut $\frac{n(n-1)}{(n-r+2)(n-r+1)}$. Finalement $C^{(1)} = O(r)$.

Étage 2 : La longueur L_2 est de 1. Il y a $\binom{n}{r-2}(n-r+2)$ transitions et $2\binom{n-2}{r-2}$ transitions où circule un flot non nul. La spécialité T_2 vaut $\frac{n(n-1)}{2(n-r+1)}$. Finalement $C^{(2)} = O\left(\sqrt{n}\right)$.

Étage 3 : La longueur L_3 est de 1. Il y a $\binom{n}{r-1}(n-r+1)$ transitions et $2\binom{n-2}{r-2}$ transitions où circule un flot non nul. La spécialité T_1 vaut $\frac{n(n-1)}{2(r-1)}$. Finalement $C^{(3)} = O\left(\frac{n}{\sqrt{r}}\right)$.

On obtient une complexité en $O\left(r + \sqrt{n} + \frac{n}{\sqrt{r}}\right)$, que l'on minimise en prenant $r = n^{\frac{2}{3}}$

Ainsi, la complexité finale est $O\left(n^{\frac{2}{3}}\right)$, qui est en fait optimale pour *element distinctness.* □

Les nombreux choix effectués au cours du calcul ont pu sembler arbitraires ; pourtant le résultat obtenu est optimal. Ceci illustre bien la grande liberté qu'octroie ce modèle ; dessiner un bon learning graph nécessite

une bonne intuition sur le problème afin de trouver les paramètres qui donneront une complexité intéressante. Par la suite nous nous servirons surtout des deux learning graph déjà introduits, la recherche de Grover et la marche sur le graphe de Johnson. Ils nous serviront d'éléments de base pour construire des learning graphs plus complexes. De même que pour Grover, on utilisera une représentation simplifiée en cachant l'arborescence. Une marche sur le graphe de Johnson sera donc représentée par trois flèches simples, une pour chaque étage de la marche.

1.5 Learning graphs et circuits électriques

Il est possible d'aborder la complexité d'un learning graph d'une autre façon. Si l'on envisage le graphe comme un réseau électrique où le poids des arêtes représente une conductance, alors en envoyant un courant d'un Ampère au sommet $\emptyset$ et en branchant les sommets de M sur la masse, le courant obtenu à travers le circuit satisfait toute les hypothèses d'un flot sur un learning graph. Sous cet angle, on constate que l'énergie du flot est en fait exactement la perte d'énergie par effet Joule dans les résistances du circuit. Or, cette énergie est naturellement minimisée dans un circuit réel. Le courant est donc directement le flot optimal pour le learning graph. On peut alors assimiler la complexité positive du learning graph à la résistance effective du circuit, qui elle peut se calculer par les méthodes usuelles sur les circuits électriques (association de résistances en série, en parallèle et équivalence triangle-étoile). Cette interprétation nous donne alors une autre méthode de calcul : on fixe les poids, puis on calcule la résistance. Si on note W la somme des poids et R la résistance, la complexité du learning graph est en $O(\sqrt{WR})$. Appliquons cette méthode au learning graph pour la recherche de Grover, mais cette fois-ci intéressons nous au cas où il y a au moins k fois l'élément recherché dans une liste de taille n.

Lemme 6 *La complexité du learning graph correspondant à une recherche de Grover sur n éléments, où l'élément recherché s'il est présent, l'est au moins k fois, a une complexité en $O\left(\sqrt{\frac{n}{k}}\right)$.*

Preuve. Toutes les arêtes ont le même poids 1, on a donc $W = n$. On choisit arbitrairement k 1-certificats branchés à la masse. Ils ont même potentiel 0, on peut alors les identifier à un seul sommet. Aucun courant ne transite à travers les arêtes qui ne mènent pas à la masse, nous n'en tenons donc pas compte dans le calcul de la résistance effective. Le graphe consiste alors en k résistances parallèles de conductance 1 chacune, ce qui est équivalent à une unique résistance de conductance k. On a $R = \frac{1}{K}$ et finalement une complexité en $O(\sqrt{\frac{n}{k}})$. □

Pour $k = 1$ on retrouve bien la recherche de Grover classique. Ce résultat, plus général, nous sera utile par la suite. Pour un graphe aussi simple que celui de Grover, cette méthode est applicable, mais elle devient vite trop calculatoire. Pour le graphe d'*element distinctness* en identifiant les sommets de même potentiel, on peut se ramener à un peu moins d'une dizaine de sommets mais le graphe demeure assez complexe pour que les calculs prennent plusieurs pages.

Cependant cette idée de circuit électrique à un intérêt théorique. Il existe une manière de concevoir des marches quantiques trouvant les sommets de M sur un graphe pondéré en $O(\sqrt{WR})$. Cette méthode s'applique aussi sur un learning graph et assure l'existence d'un algorithme quantique de même complexité que le learning graph justifiant l'utilisation du modèle pour calculer des complexités.

2 Application à la recherche de triangle

2.1 Le problème de recherche de triangle

Le problème de la recherche de triangle est l'un des plus simple de la théorie des graphes, et bien que ses applications soient nombreuses en étude des réseaux, sa complexité quantique exacte demeure inconnue à ce jour.

Soit un graphe $G = (V, E)$ d'ordre n. On cherche à savoir si celui-ci contient un triangle, c'est-à-dire trois sommets tous reliés entre eux. Pour ce problème, une requête consiste à interroger une paire de sommets pour savoir s'ils sont reliés par une arrête ou non. L'ensemble des requêtes possibles est donc de taille $\frac{n(n-1)}{2}$. Un 1-certificat pour ce problème consiste en un ensemble de sommets a, b et c tels que ab, bc et $cd \in E$. Appliquer de manière triviale la recherche de Grover sur l'ensemble des triangles potentiels donne une borne supérieure sur la complexité en $O(n^{1.5})$. La meilleure borne inférieure connue s'obtient par réduction à partir de la recherche d'éléments dans une liste.

Lemme 7 *La compléxité quantique de la recherche de triangle dans un graphe de taille n est en $\Omega(n)$.*

Preuve. Soit une liste de n éléments. On l'encode dans les arêtes d'un graphe : une paire de sommets est une arête si et seulement si elle correspond à l'élément recherché. Aux sommets représentant la liste, on ajoute un sommet supplémentaire relié à tous les autres. Le graphe obtenu possède alors $O(\sqrt{n})$ sommets et ne contient un triangle que si la liste contient l'élément recherché. Or on sait que l'algorithme de Grover en $O(\sqrt{n})$ est optimal pour la recherche d'élément dans une liste. Il en suit que l'on ne peut trouver un triangle en moins de $O(n)$ requêtes, où n est l'ordre du graphe. □

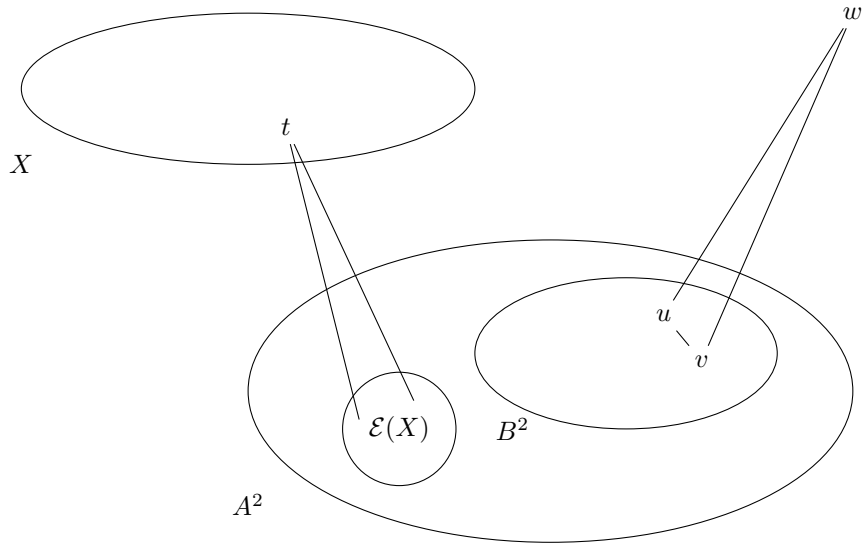
On sait cependant qu'aucun learning graph non-adaptatif ne peut faire mieux que $O(n^{\frac{9}{7}})$, borne atteinte par Lee, Magniez et Santha [9]. En utilisant des arguments combinatoires, Le Gall a obtenu un algorithme en $O(n^{\frac{5}{4}} \log(n))$ [11].

2.2 Algorithme de Le Gall

Nous ne détaillerons pas l'algorithme au point d'expliciter précisément sa complexité mais simplement assez pour avoir une idée générale de son fonctionnement afin de construire ensuite un learning graph adapté. L'idée principale de l'algorithme de Le Gall est d'exclure un ensemble X de sommets afin de limiter le nombre de requêtes à effectuer.

En pratique, on commence par choisir aléatoirement un ensemble X, de taille x suffisamment petite, pour qu'y rechercher un sommet de triangle en utilisant une recherche de Grover ne soit pas trop coûteux. Si on trouve un triangle, on arrête là . Sinon, on dispose d'un ensemble de sommets qui à coup sûr n'appartiennent à aucun triangle. A partir de X, on définit $\mathcal{E}(X)$, l'ensemble des couples de sommets qui sont tout deux voisins d'un même sommet de X. Les propriétés de X nous permettent d'assurer que $\mathcal{E}(X)$ ne contient aucune arête. Cependant, déterminer $\mathcal{E}(X)$ en entier risque de nécessiter trop de requêtes, alors on se limite à un

ensemble A de taille a que l'on recherche par marche sur le graphe de Johnson. On détermine alors $\mathcal{E}(X) \cap A^2$. Maintenant, on recherche avec Grover un sommet de triangle w. Toujours pour économiser des requêtes, on ne recherche pas directement l'arête uv qui forme le triangle avec w. On se limite à un ensemble B de taille b dans A que l'on recherche par une marche sur le graphe de Johnson. Il ne reste alors qu'à rechercher dans B^2 l'arête uv. Cependant, nous savons qu'il ne sert à rien de questionner les paires de $\mathcal{E}(X) \cap A^2$; on note $\Delta(X, B)$ l'ensemble des paires candidates. En fait, on a $\Delta(X, B) = B^2 \backslash \mathcal{E}(X)$. De plus on ne recherche que parmi les paires qui peuvent former un triangle avec w. On note l'ensemble de ces paires $\Delta(X, B, w)$; on a $\Delta(X, B, w) = (B \cap N(W))^2 \cap \mathcal{E}(X)$. L'algorithme se termine par une recherche de Grover sur $\Delta(X, B, w)$.

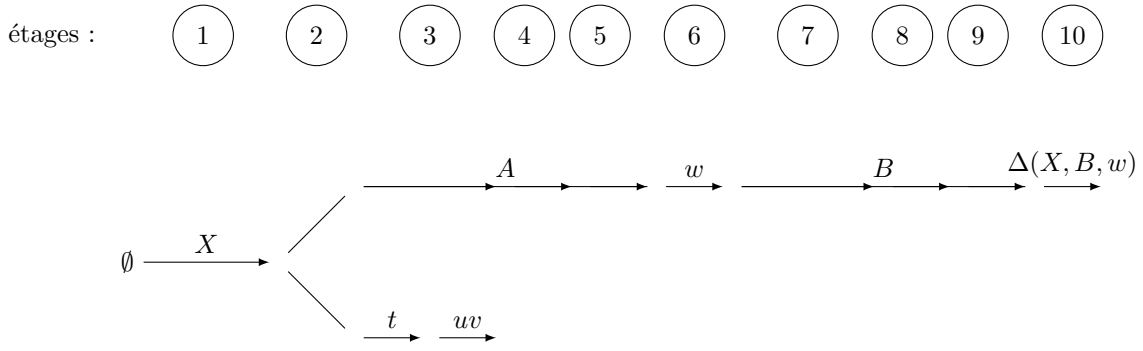


2.3 Algorithme à base de Learning graphs

On remarque que l'on peut voir l'algorithme de Le Gall comme un enchaînement de recherches de Grover et de marches sur le graphe de Johnson. Or nous connaissons bien les learning graphs associés à ces routines. On peut alors en combinant ces éléments obtenir un learning graph utilisant les idées de l'algorithme.

2.3.1 Mise sous forme de learning graph

On va maintenant utiliser le formalisme des learning graphs pour concevoir un algorithme efficace pour la recherche de triangles. Pour ce faire, on découpe l'algorithme en sous-routines que l'on branchent à la suite les unes des autres dans le learning graph. En utilisant les conventions de représentation introduites, nous obtenons un learning graph de cette forme :



Il s'agit d'un learning graph à 10 étages dont le dernier est adaptatif. L' étage 1 consiste à considérer tout les ensembles X possibles à la manière d'une recherche de Grover. Dans l'étage 2, il y a pour chaque X un embranchement, l'un correspondant à la recherche de sommets de triangle dans X et l'autre à la recherche de triangles, en admettant que X a les bonnes propriétés. Suivant le X le flot n'ira que dans l'une des deux branches. Les étages 3, 4 et 5 constituent la marche sur le graphe de Johnson pour trouver A. L'étage 6 est une recherche de Grover sur tous les w. Les étages 7, 8 et 9 correspondent à la marche sur le graphe de Johnson pour trouver B, et enfin l'étage 10 est la recherche de Grover sur $\Delta(X, B, W)$ dont la taille varie selon l'entrée : c'est là qu'apparaît le caractère adaptatif du learning graph.

2.3.2 Quelques lemmes

Le learning graph obtenu est imposant et surtout adaptatif, il va donc nous falloir modifier les méthodes non-adaptatives et les lemmes suivants vont grandement simplifier l'analyse. Ces lemmes sont très généraux et peuvent être utiles dans des cas bien différents de l'algorithme qui nous intéresse ici.

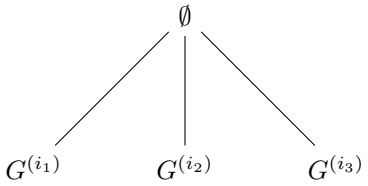
Learning graph pour la disjonction

Lemme 8 *Soit une famille de fonctions f_i de complexité en terme de learning graph en $O(C^{(i)})$ et h la fonction telle que $h = \bigvee_i f_i$. Alors la complexité en terme de learning graph de h est en $O\left(\sqrt{\sum_i (C^{(i)})^2}\right)$. De plus si l'on a l'assurance que si $h(x) = 1$ alors $f_i(x) = 1$ pour au moins k des fonctions f_i alors la complexité est en $O\left(\sqrt{\frac{\sum_i (C^{(i)})^2}{k}}\right)$.*

Nous allons montrer le lemme dans un cadre un peu plus général que celui où il est énoncé. En effet l'intérêt de ce lemme est de décomposer des learning graphs en assimilant les sous-parties aux fonctions qu'elles calculent. Mais dans le cas adaptatif, ces learning graphs dépendent de l'entrée. Cette idée de dépendance est logique en terme de learning graphs mais étrange en terme de fonctions. Ainsi, afin d'éviter les ambiguïtés, nous allons passer par un lemme intermédiaire.

Afin de simplifier les écritures, on définit les *transitions vides* ; qui sont des arêtes factices qu'il faut contracter (au sens des graphes) pour obtenir un vrai learning graph. On peut les voir comme des fils sans résistance dans l'interprétation électrique.

Lemme 9 *Soit un learning graph G de la forme suivante :*



Le sommet $\emptyset$ est relié par des transitions vides à des composantes indépendantes, c'est à dire qu'aucune autre arête ne les relie entres elles. On appelle s_i les sommets reliés à $\emptyset$ et $G^{(i)}$ les learning graphs correspondants aux composantes où s_i est assimilé à $\emptyset$. On note $C^{(i)}$ leurs complexités. Alors la complexité de G est en $O\left(\sqrt{\sum_i (C^{(i)})^2}\right)$. De plus, si des sommets marqués sont atteints par le flot dans au moins k composantes, la complexité est en $O\left(\sqrt{\frac{\sum_i (C^{(i)})^2}{k}}\right)$.

Preuve. Pour montrer ce lemme, on calcule simplement la complexité de G. On choisit arbitrairement k composantes contenant un sommet marqué. On choisit de router le flot de manière uniforme à travers les transitions vides menant à ces composantes. Chacune d'elles reçoit donc $\frac{1}{k}$ unités de flot. A l'intérieur des composantes $G^{(i)}$ on route le flot de manière optimale pour atteindre la complexité $C^{(i)}$. On a alors, en multipliant dans chaque composante tous les poids par la complexité positive du learning graph associé :

$$C_0 = \sum_i C_0^{(i)} C_1^{(i)} = \sum_i (C^{(i)})^2 \tag{2.1}$$

$$C_1 = \sum_{i=1}^{k} \frac{1}{k^2} = \frac{1}{k} \tag{2.2}$$

D'où le résultat. □

On déduit alors directement le lemme 8 du lemme 9, en constatant qu'une structure de certificat h est l'union des structures de certificats des f_i. Il en suit qu'un learning graph constitué du sommet $\emptyset$ relié par des transitions vides à des learning graphs optimaux pour les f_i est un learning graph valide pour h. En appliquant le lemme 9, on obtient le résultat souhaité.

Spécialité adaptative

Lemme 10 *Soit un étage de transitions parallèles regroupées en k classes $\mathcal{H}_i$, où k est une constante, de mêmes longueurs L_i, et où chaque classe respecte les hypothèses d'utilisation de la spécialité. On suppose que le flot prend seulement deux valeurs dans tout l'étage : 0 et une constante c. De plus, on suppose le flot uniforme sur toutes les transitions où il n'est pas nul ; ainsi, si l'étage reçoit f unités de flot, alors chaque transition utile en reçoit une part égale. La complexité de l'étage est alors en $O\left(M_q(L)\sqrt{T(x)}\right)$ où $T(x)$ est la spécialité adaptative de l'étage et $M_q(L)$ la moyenne quadratique des longueurs des transitions de l'étage.*

On retrouve bien le cas classique lorsque $k = 1$.

Preuve. Pour montrer cela il suffit de donner à toutes les arêtes d'une même classe un même poids w_i. On introduit le nombre de transitions dans une classe n_i et le nombre de transitions utiles par classe n_i^{utile}. On a donc $n_{\text{utile}} = \sum_i n_i^{\text{utile}}$ et $n_{\text{tot}} = \sum_i n_i$. On pose $w_i = \frac{L_i}{n_{\text{tot}}}$. On a alors :

$$C_0 = \sum_{i=1}^{k} L_i w_i n_i = (M_q(L))^2 \tag{2.3}$$

$$C_1 = \sum_{i=1}^{k} n_i^{\text{utile}} L_i \frac{c^2}{w_i n_{\text{utile}}^2} = c^2 n_{\text{tot}} \sum_{i=1}^{k} \frac{n_i^{\text{utile}}}{n_{\text{utile}}^2} = f^2 T(x) \tag{2.4}$$

or $f \leq 1$ d'où le résultat. □

Ici les poids originaux et la spécialité peuvent dépendre de l'entrée, par contre le flot doit toujours satisfaire les hypothèses pour toute entrée.

2.3.3 Calcul de la complexité

Nous allons calculer étage par étage en utilisant abondamment les lemmes précédents. Pour faciliter l'utilisation du lemme 9 nous appellerons $G(X)$ le learning graph correspondant à l'exécution de l'algorithme une fois un X fixé, G_1 le learning graph correspondant à la recherche de sommets de triangle dans X (on remarque qu'il n'est pas adaptatif), et $G_2(X)$ le learning graph associé à la recherche de triangles avec un X convenable. On note $C(X)$, C_1 et $C_2(X)$ leur complexités respectives. On rappelle que l'on a la hiérarchie

suivante sur les tailles des différents ensembles : $b = o(a)$ et $a = o(n)$.

Étage 1 : Il s'agit de transitions vides représentant les différents X, nous sommes dans le cas du lemme 9. Chaque branche recevant un flot uniforme, tous les X mènent à une exécution correcte de l'algorithme, on a une complexité pour le learning graph en $O\left(\sqrt{\frac{\sum_X C(X)^2}{\binom{n}{x}}}\right)$.

Étage 2 : On symbolise par des transitions vides l'embranchement vers $G_1(X)$ et $G_2(X)$. L'application du lemme 9 nous donne une complexité $C(X)$ en $O(\sqrt{C_1^2 + C_2(X)^2})$. Calculons C_1. Ce learning graph est un embranchement de transitions vides représentant chaque $t \in X$ suivi d'une recherche de Grover sur les couples de sommets du graphe qui formeraient un triangle avec ce t. Cette dernière recherche est en $O(n)$ et en appliquant le lemme 9 on trouve une complexité C_1 en $O(n\sqrt{x})$.

Étage 3 : Dorénavant tous les calculs ont pour objectif de déterminer $C_2(X)$, nous ne ferons usage que de la spécialité adaptative. Tous les étages suivants sont constitués de transitions de mêmes longueurs. Nous choisissons, comme depuis le début de ce rapport, de ne considérer que les 1-certificat correspondant à un unique triangle, composé d'une arête uv et d'un troisième sommet w. On commence par chercher A de taille a. Dans cet étage on ne route le flot qu'à travers les transitions menant aux ensembles de taille $a-2$ ne contenant pas uv. Attention : le but étant ici de calculer $\mathcal{E}(X) \cap A^2$, on calcule les voisins de chaque élément dans X. Ainsi, ajouter un élément à A coûte en fait x requêtes. La longueur L_3 de l'étage est donc $(a-2)x$, il y a $\binom{n}{a-2}$ transitions au total et $\binom{n-2}{a-2}$ transitions utiles, d'où une spécialité T_3 en $O(1)$. Finalement, la complexité de cet étage est en $O(ax)$.

Étage 4 : On route le flot à travers les transitions qui chargent le premier sommet de l'arête uv. La longueur L_4 est de x, il y a $\binom{n}{a-2}(n-a+2)$ transitions au total et $\binom{n-2}{a-2}$ transitions utiles, d'où une spécialité en $O(\sqrt{n})$. La complexité de l'étage est en $O(x\sqrt{n})$.

Étage 5 : On route le flot à travers les transitions qui chargent le dernier sommet de l'arête uv. La longueur L_5 est de x, il y a $\binom{n}{a-1}(n-a+1)$ transitions au total et $\binom{n-2}{a-2}$ transitions utiles, d'où une spécialité en $O(\frac{n^2}{a})$. La complexité de l'étage est en $O(\frac{xn}{\sqrt{a}})$.

Étage 6 : Il s'agit de transitions vides représentant une recherche de Grover sur les w. Plutôt qu'utiliser le lemme 9, nous nous rappellerons par la suite que pour chaque A contenant uv on route le flot dans une unique transition correspondant au bon w parmi les n possibilités. Aucune requête n'est effectuée à cet étage.

Étage 7 : On débute maintenant la recherche de B dans A tel que B contienne uv. A chaque fois qu'on charge un élément dans B on vérifie s'il est voisin de w ce qui coûte une requête. On ne route le flot que dans les transitions menant aux ensembles ne contenant pas uv. La longueur L_7 est de $b-2$, il y a $\binom{n}{a}n\binom{a}{b-2}$ transitions au total et $\binom{n-2}{a-2}\binom{a-2}{b-2}$ transitions utiles, d'où une spécialité en $O(\frac{n^3}{a^2})$. La complexité de l'étage est en $O(\frac{bn^{\frac{3}{2}}}{a})$.

Étage 8 : On route le flot à travers les transitions chargeant le premier sommet de uv. La longueur L_8 est de 1, il y a $\binom{n}{a}n\binom{a}{b-2}(a-b+2)$ transitions au total et $\binom{n-2}{a-2}\binom{a-2}{b-2}$ transitions utiles, d'où une spécialité en $O(\frac{n^3}{a})$. La complexité de l'étage est en $O(\frac{n^{\frac{3}{2}}}{\sqrt{a}})$.

Étage 9 : On route le flot à travers les transitions chargeant le second sommet de uv. La longueur L_9 est de 1, il y a $\binom{n}{a}n\binom{a}{b-1}(a-b+1)$ transitions au total et $\binom{n-2}{a-2}\binom{a-2}{b-2}$ transitions utiles, d'où une spécialité en $O(\frac{n^3}{b})$. La complexité de l'étage est en $O(\frac{n^{\frac{3}{2}}}{\sqrt{b}})$.

Étage 10 : On fini en interrogeant l'arête uv elle même, par une recherche de Grover sur $\Delta(X,B,w)$. La longueur L_{10} est de 1, il y a $\sum_{A,B,w} |\Delta(X,B,w)|$ transitions au total et $\binom{n-2}{a-2}\binom{a-2}{b-2}$ transitions utiles, d'où une spécialité en $O\left(\frac{\sum_{A,B,w}|\Delta(X,B,w)|}{\binom{n-2}{a-2}\binom{a-2}{b-2}}\right)$. La complexité du dernier étage est en $O\left(\sqrt{\frac{\sum_{A,B,w}|\Delta(X,B,w)|}{\binom{n-2}{a-2}\binom{a-2}{b-2}}}\right)$.
Pour simplifier l'écriture, on fait apparaître la moyenne sur B et w de $|\Delta(X,B,w)|$ que l'on note $M_{B,w}(X)$. On obtient une complexité en $O(\frac{n^{\frac{3}{2}}}{b}\sqrt{M_{B,w}(X)})$.

On obtient finalement une complexité $C_2(X)$ en $O(ax + x\sqrt{n} + \frac{xn}{\sqrt{a}} + \frac{bn^{\frac{3}{2}}}{a} + \frac{n^{\frac{3}{2}}}{\sqrt{a}} + \frac{n^{\frac{3}{2}}}{\sqrt{b}} + \frac{n^{\frac{3}{2}}}{b}\sqrt{M_{B,w}(X)})$.

On choisit de prendre $a = n^{\frac{3}{4}}$, $b = \sqrt{n}$ et $x = \sqrt{n}$.

Ce qui nous donne : $C_2(X) = O(n^{\frac{5}{4}} + n + n^{\frac{9}{8}} + n + n^{\frac{9}{8}} + n^{\frac{5}{4}} + n\sqrt{M_{B,w}(X)})$.

On a alors $C_1 = O(n^{\frac{5}{4}})$ et $C_2(X) = O(n^{\frac{5}{4}} + n\sqrt{M_{B,w}(X)})$.

Ainsi $C(X) = O(n^{\frac{5}{4}} + n\sqrt{M_{B,w}(X)})$ et donc la complexité finale est en $O(\sqrt{n^{\frac{5}{4}} + n^2 M_{X,B,w}})$ où $M_{X,B,w}$ est la moyenne sur les X, B et w de $|\Delta(X,B,w)|$. Sous cette forme la complexité dépend encore de l'entrée mais le lemme suivant va nous permettre de supprimer cette dépendance.

Lemme 11 *Pour tout graphe $G = (V,E)$ de taille n, pour tout $B \subset V$ de taille b, on a $M_{X,B,w} \leq \frac{b^2}{x}$, où x est la taille de X.*

Preuve. Soit $(u,v) \in V^2$. On note $T_{u,v}$ l'ensemble $N(u) \cap N(v)$ de taille t et $\mathcal{P}((u,v) \in \Delta(X,B,w))$ la probabilité que (u,v) appartienne à $\Delta(X,B,w)$. On note aussi $\Delta(X)$ l'ensemble des couples de sommets qui ne sont pas tous deux voisins d'un même z dans X.
Soit $B \subset V$. Calculons l'espérance sur X et w :

$$E_{X,w}[|\Delta(X,B,w)|] = \sum_{(u,v)\in B^2} \mathcal{P}((u,v) \in \Delta(X,B,w)) \tag{2.5}$$

D'autre part on a :

$$\mathcal{P}((u,v) \in \Delta(X,B,w)) = \mathcal{P}((u,v) \in \Delta(X) \text{ et } w \in T_{u,v}) \tag{2.6}$$

Or, comme $w \notin \Delta(X)$, ces deux événement sont indépendants.

$$\mathcal{P}((u,v) \in \Delta(X,B,w)) = \frac{t}{n}\left(1 - \frac{t}{n}\right)^x \tag{2.7}$$

En posant $\alpha = \frac{tx}{n}$ on a :

$$\mathcal{P}((u,v) \in \Delta(X,B,w)) = \frac{\alpha}{x}\left(1 - \frac{\alpha}{x}\right)^x \leq \frac{\alpha e^{-\alpha}}{x} \leq \frac{1}{x} \tag{2.8}$$

Et finalement :

$$E_{X,w}[|\Delta(X,B,w)|] \leq \frac{b^2}{x} \tag{2.9}$$

□.

Dans le cas qui nous intéresse, on a donc $M_{X,B,w} \leq \sqrt{n}$, soit une complexité pour notre learning graph en $O(n^{\frac{5}{4}})$.

Nous pouvons donc transformer ce learning graph en un algorithme quantique qui décide l'existence d'un triangle dans un graphe en $O(n^{\frac{5}{4}})$ requêtes améliorant d'un facteur logarithmique l'algorithme de Le Gall.

Conclusion

Dans ce rapport, on construit un learning graph adaptatif pour le problème de la détection de triangles dans un graphe avec une complexité en $O(n^{\frac{5}{4}})$. Il peut être transformé en algorithme quantique de même complexité. On améliore ainsi d'un facteur logarithmique la meilleure borne supérieure connue sur la complexité en requête de ce problème. Pour ce faire, on a généralisé au cas adaptatif des méthodes usuelles d'analyse des learning graphs. On trouve, pour l'instant, peu de learning graphs adaptatifs en tant que tels dans la littérature ; on les sait pourtant strictement plus puissants que leurs homologues non adaptatifs, et il serait intéressant d'appliquer ces raisonnements à des problèmes proches de la détection de triangles comme par exemple *associativity testing*, chose que je n'ai pu approfondir au cours de ce stage. Il n'y a pour l'instant, à ma connaissance, pas de meilleure borne inférieure pour les learning graphs adaptatifs que $\Omega(n)$; on ne sait donc pas si ce learning graph adaptatif est optimal ou non.

Le principal intérêt théorique de ces techniques demeure, plus que la recherche d'algorithmes pouvant être implémentés sur d'éventuels ordinateurs quantiques, la compréhension du gain de complexité qu'offrirait un tel ordinateur. C'est un sujet de recherche très actif et c'est toujours une question ouverte que de connaître exactement les écarts possibles entre les complexités en requête classique, probabiliste et quantique.

Bibliographie

[1] A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3) : 786–807, 2010.

[2] A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. *Proc. of the 53rd IEEE FOCS*, 2012.

[3] A. Belovs. Span programs for functions with constant-sized 1-certificates. *Proc. of 44th ACM STOC*, 2012.

[4] A. Belovs. Quantum walks and electric networks. 2013.

[5] A. Belovs. Applications of the adversary method in quantum query algorithms. Doctoral thesis, 2013.

[6] A. Belovs et T. Lee. Quantum algorithm for k-distinctness with prior knowledge on the input. 2011.

[7] A. M. Childs, notes de cours, Quantum algorithms (CO 781/CS 867/QIC 823, University of Waterloo, Winter 2013), Adversary method et Learning graphs. 2013.

[8] Y. Eichenlaub et P. Lagonotte. Réseaux électrocinétiques et algèbre linéaire (notions fondamentales). TP, I.U.T. de Poitiers, Département GEII.

[9] T. Lee, F. Magniez et M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. *Proc. of the 24th ACM-SIAM SODA*, 2013.

[10] D. A. Levin, Y. Peres et E. L. Wilmer. Markov chains and mixing times. Version électronique, *pages.uoregon.edu/dlevin/, Chapitre 9*. 2008.

[11] F. Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. *Proc. of the 55th IEEE FOCS*, 2014.

[12] M. Santha. Quantum walk based search algorithms. *Proc. of 5th TAMC*, volume 4978 of LNCS, pages 31–46. Springer, 2008.