Algorithmes de recherche et problème des collisions en calcul quantique

Pierre Philipps

Rapport du stage effectué au LRI-Orsay
du 1^{er} Juin au 13 Juillet 2001

Introduction

En 1982, Richard Feynman a remarqué que l'on ne pouvait pas simuler de manière efficace certains effets quantiques sur un ordinateur classique[7], d'où l'idée d'utiliser ces effets pour effectuer des calculs de manière plus efficace. Ce n'est qu'en 1994 que Peter Shor, en publiant un algorithme quantique de factorisation d'entiers en temps polynômial, a montré l'intérêt d'une telle approche de l'informatique[8]. En effet, le meilleur algorithme classique connu est de complexité exponentielle (Lenstra et Lenstra 1993). Par ailleurs, Lov Grover a présenté en 1996 un algorithme de recherche d'un élément dans une liste non structurée de n objets avec une complexité de $O(\sqrt{n})$[9]. Cet algorithme est plus efficace que n'importe quel algorithme classique; on sait en effet que la recherche d'un élément dans une liste non structurée se fait sur un ordinateur classique en temps linéaire.

Malgré ces résultats positifs, la question de la viabilité d'un ordinateur quantique demeure. En effet, le plus gros ordinateur quantique construit jusqu'à présent ne manipule que 5 qbits (l'équivalent quantique des bits classiques). De plus, maintenir une superposition d'états quantiques est difficile à cause des interactions possibles avec l'environnement. Cependant, l'apparition récente de codes correcteurs d'erreurs quantiques permet d'espérer contourner ce genre de problèmes.

Toutes ces raisons font que la recherche en calcul quantique est un sujet d'actualité en plein développement. L'objectif du stage était d'étudier les principes généraux de calcul quantique puis de se pencher plus particulièrement sur l'algorithme de Grover et les problèmes de collisions. Ce rapport présente notamment les apports suivants : tout d'abord un algorithme de recherche exact uniforme dans une liste non structurée de complexité $O(N^{\frac{\log 3}{\log 4}})$, ensuite une borne inférieure dans un cas très particulier du problème de décision de l'existence d'une solution à l'équation $f(x) = 1$ et enfin un nouvel algorithme de résolution du problème des k-collisions de complexité $O(N^{1-\frac{1}{2^k}})$.

1 Modèle de calcul quantique

Nous nous intéressons dans cette partie à une présentation du modèle de calcul quantique d'un point de vue uniquement théorique. En particulier, cette présentation n'inclut pas l'explication physique de l'origine des contraintes que l'on impose à notre système de calcul.

1.1 L'unité de base de calcul : le qbit

1.1.1 1 qbit

Un bit classique est simplement une valeur de vérité : vrai ou faux, 1 ou 0. Un bit quantique, que l'on nomme qbit, est une superposition quantique de 0 et de 1; c'est en fait un vecteur normé d'un $\mathbb{C}$-espace vectoriel de dimension 2 dont 0 et 1 forment une base orthonormale. En calcul quantique (de même qu'en mécanique quantique de manière générale), on utilise la notation bra-ket; ainsi, le vecteur 0 se note $|0\rangle$. Si $|\psi\rangle$ est un qbit, il existe donc deux complexes a et b tels que :

$$|\psi\rangle = a|0\rangle + b|1\rangle \text{ et } |a|^2 + |b|^2 = 1$$

La seule manière d'accéder à la valeur d'un qbit est d'effectuer ce qu'on appelle une mesure de ce qbit. Le résultat de la mesure est alors $|0\rangle$ avec la probabilité $|a|^2$ et $|1\rangle$ avec la probabilité $|b|^2$. De plus, après la mesure, l'état du qbit est modifié et est alors égal au résultat de la mesure (c'est-à-dire $|0\rangle$ si l'on a mesuré $|0\rangle$).

1.1.2 n qbits, ou 1 n-qbit

L'intérêt d'utiliser des effets quantiques commence à apparaître lorsque l'on ne considère non plus un seul qbit, mais un ensemble de n qbits. En effet, contrairement au cas classique où un ensemble de n bits est entièrement décrit par chacun des bits séparèment, ces n qbits peuvent être corrélés d'une manière que nous allons maintenant exposer.

Dans le paradoxe EPR (Einstein, Podolsky et Rosen), deux particules (ou qbits) sont générées par une source de telle manière qu'elles sont nécessairement dans le même état. Alors, quelle que soit la manière, le moment ou l'endroit où l'on effectue une mesure, le résultat de cette mesure sera le même pour les deux particules. Cependant, ce résultat peut être arbitrairement $|0\rangle$ ou $|1\rangle$ et des considérations de mécanique quantique montrent que l'on ne peut considérer que les deux particules étaient toutes deux au moment de leur création soit dans l'état $|0\rangle$, soit dans l'état $|1\rangle$, et ce jusqu'à la mesure. Il faut donc en déduire que les deux particules étaient chacune dans une superposition des deux états, mais de telle sorte que la mesure effectuée sur l'une des deux affecte l'autre de telle manière qu'elle se retrouve alors inévitablement dans l'état mesuré sur la première.

Pour expliquer ce qui se passe, il faut admettre que l'on ne peut considérer chacun des deux qbits indépendament; il faut plutôt considérer les 2 qbits comme un système global, que l'on nomme un 2-qbit. Ce système est alors dans une superposition des états $|00\rangle$ et $|11\rangle$. Plus précisèment, il est représenté par un vecteur normé d'un $\mathbb{C}$-espace vectoriel de dimension 4, dont une base orthonormale est la suivante : $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$. Dans le cas du paradoxe EPR, on a donc la superposition suivante : $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, ce qui fait que lorsque l'on mesure l'un des deux qbits dans l'état $|0\rangle$ ou $|1\rangle$, l'autre se retrouve dans le même état.

De manière générale, on définit un n-qbit comme un vecteur normé d'un $\mathbb{C}$-espace vectoriel de dimension 2^n. Par la suite, un état de la base de mesure ne sera plus représenté par une succession de 0 et de 1, mais par un nombre compris entre 0 et $2^n - 1$. Par exemple pour 2 qbits, la base de travail est $(|0\rangle, |1\rangle, |2\rangle, |3\rangle)$.

1.2 Porte quantique

Nous avons donc jusqu'à présent défini ce que sont les bits quantiques; nous allons voir maintenant comment les manipuler. Nous abordons ici le problème en termes de circuits. Il faut donc définir un certain nombre de portes que nous aurons le droit d'utiliser pour fabriquer un algorithme quantique. La règle principale est que les transformations effectuées sur les qbits doivent être réversibles. Comme elles doivent de plus conserver la norme des vecteurs considérés (plus précisèment, ces derniers doivent rester normés au cours de l'opération), on impose donc que les transformations utilisées soient unitaires. Ceci, et plus particulièrement la linéarité des transformations utilisées impose notamment qu'il est impossible en calcul quantique de clôner directement un qbit.

Un exemple important de transformation utilisée est la porte d'Hadamard. Elle agit sur un qbit et est définie par la matrice suivante :

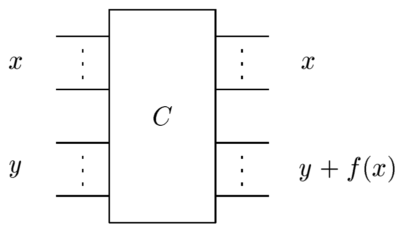
$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

La porte d'Hadamard a la particularité de transformer le qbit $|0\rangle$ en une superposition uniforme de $|0\rangle$ et $|1\rangle$.

A partir de cette porte, on définit la porte de Walsh-Hadamard sur un n-qbit, qui consiste à appliquer simultanément la transformation d'Hadamard sur chacun des n qbits qui forment ce n-qbit. De manière similaire à la porte d'Hadamard simple, la porte de Walsh-Hadamard transforme le n-qbit $|0\rangle$ en une superposition uniforme des 2^n états de la base de mesure de ce n-qbit. Nous appelerons par la suite cette porte H.

1.3 Modèle de calcul d'une fonction

Maintenant, il faut définir ce que cela signifie qu'un circuit quantique calcule une fonction f. Comme on ne peut imposer à toutes les fonctions que l'on calcule d'être réversibles, ce qui restreindrait énormèment les possibilités, on ne peut envisager une solution où la sortie se lirait directement à la sortie du circuit. On utilise donc la technique suivante pour contourner cette difficulté : on se donne une fonction f quelconque de $\{0,\ldots,2^n-1\}$ dans $\{0,\ldots,2^m-1\}$. On dira que le circuit quantique C sur $n+m$ qbits calcule f si et seulement s'il vérifie le shéma suivant (avec $x \in \{0,\ldots,2^n-1\}$ et $y \in \{0,\ldots,2^m-1\}$) :



Ici, dans $y + f(x)$, le $+$ est en fait un ou exclusif bit à bit. Il est facile de vérifier que le circuit C ainsi défini est unitaire pour toute fonction f. Cependant, cette définition ne fonctionne que dans le cas où l'ensemble de départ de la fonction est fini. Comment définir en calcul quantique le fait de calculer une fonction dont l'ensemble de départ est infini, ou bien encore une famille infinie de fonctions? Tout ceci revient finalement à définir le fait de résoudre un problème en calcul quantique (par exemple, un problème de tri, ou de recherche). Pour ce faire, il faut considérer un ensemble de circuits $(C_n)_{n\in\mathbb{N}}$, où n est la taille de l'entrée. On appelle alors porte de base de cette famille de circuit toute porte utilisée dans la définition de ces circuits. Cependant, on impose généralement une condition supplémentaire à cette famille de circuits : elle doit être uniforme, c'est-à-dire qu'il doit exister un algorithme (classique) qui construit le circuit C_n lorsque l'on lui donne n comme entrée. Ceci impose notamment que les portes de base utilisées dans tous ces circuits soient en nombre fini, ce qui n'empêche pas de les utiliser plusieurs fois chacune.

Par ailleurs, comme l'observation d'un état quantique fait intervenir du hasard, il est naturel de s'intéresser à des algorithmes probabilistes : on dit qu'un circuit quantique calcule une fonction (ou bien qu'une famille de circuits résout un problème) avec une erreur bornée si les conditions ci-dessus pour chaque entrée sont vérifiée avec une probabilité au moins $1/2+\epsilon$ après mesure, où $\epsilon > 0$ et ϵ indépendant de l'entrée.

1.4 qbits internes

Les circuits que nous verrons par la suite pourront également utiliser, en plus des qbits d'entrée, des qbits supplémentaires appelés qbits internes; ils serviront à effectuer les calculs intermédiaires nécessaires au fonctionnement de l'algorithme. Par opposition, on pourra appeler les qbits d'entrée qbits externes. Cependant, il est un piège dangereux à éviter lors de l'utilisation de qbits internes : ces qbits peuvent subir toutes les modifications que l'on souhaite lors de l'exécution de l'algorithme, mais ils doivent impérativement être remis dans leurs états initiaux à la fin de l'algorithme. En effet, si tel n'était pas le cas, ils se trouveraient corrélés à des qbits externes; l'état de ces derniers dépendrait alors de l'état des qbits internes, ce qui fait qu'il n'y aurait plus d'interférence possible pour les qbits externes par la suite.

1.5 Simulation de calcul classique et universalité

Nous avons maintenant un résultat essentiel, dû à Deutsh(1985) : ce modèle de calcul quantique permet de simuler n'importe quel circuit classique, c'est-à-dire qu'il est possible de construire un circuit quantique pour toute fonction calculable de manière classique. Une des façons d'obtenir ce résultat ne fait intervenir que deux portes quantiques :

- la porte controlled-NOT, notée C_{NOT}. Cette porte agit sur 2 qbits et est définie sur la base classique de la manière suivante : elle inverse le second bit si le premier est à 1 et le laisse inchangé sinon; la matrice de C_{NOT} est donc la suivante :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- la porte de Toffoli, notée T. T agit de manière similaire à C_{NOT}, mais sur 3 qbits : elle inverse le dernier bit si les deux premiers sont à 1 et le laisse inchangé sinon.

Un autre problème que l'on peut se poser sur ce modèle de calcul quantique est celui d'ensemble universel de portes. Il s'agit de trouver un ensemble si possible restreint de portes quantiques qui permet de construire n'importe quel transformation unitaire. Ce problème a été résolu en 1995[10] : il suffit d'utiliser C_{NOT} ainsi que toutes les transformations unitaires sur un qbit. Remarquons que les portes utilisées dans cet ensemble universel n'agissent que sur 1 ou 2 qbits. Ce résultat est assez surprenant, puisque dans les modèles classiques de circuits réversibles, on peut montrer qu'il est nécessaire d'avoir dans un ensemble universel au moins une porte agissant sur 3 bits. On note également que l'ensemble universel proposé ci-dessus est infini; il est cependant possible de construire un ensemble fini de portes pouvant simuler de manière approchée n'importe quelle porte quantique.

1.6 Complexité

La manière la plus simple de mesurer la complexité d'une famille de circuit quantique est de majorer le nombre de portes utilisées en fonction de la taille de l'entrée. Cette complexité, que l'on appelle complexité en nombre de portes, dépend évidemment des portes de base que l'on s'autorise à utiliser. On peut par exemple se limiter aux éléments de l'ensemble universel décrit précédemment. Il est alors également possible d'utiliser des portes simples comme T ou H sans changement de complexité, car ces dernières se construisent à partir d'un certain nombre constant de portes de base.

Cependant, dans les problèmes que nous allons aborder, nous utiliserons souvent un oracle f. Les circuits décrits feront alors appel à des évaluations de f. Il est donc possible de compter plutôt le nombre d'appel à f en fonction de la taille de l'entrée. C'est ce qu'on appelle la complexité en nombre de requêtes.

1.7 Mesures en cours de calcul

Le résultat énoncé dans ce paragraphe est dû à Aharonov, Kitaev et Nisan (2001)[3]. Remarquons tout d'abord que dans le modèle de calcul que nous venons de présenter, nous n'avons à aucun moment mentionné la possibilité de faire une mesure de l'état quantique dans lequel se trouve le système; en effet, une mesure est non réversible, ce qui n'entre pas dans le cadre du modèle. On est donc amené à effectuer une seule mesure à la fin de l'algorithme. Cependant, il peut être intéressant pour des raisons de commodités de vouloir effectuer une mesure en cours de calcul. C'est pourquoi Aharonov, Kitaev et Nisan ont montré que les deux modèles de calculs (avec ou sans mesures en cours de calcul) étaient polynômialement équivalents en terme de puissance de calcul. Par conséquent, on s'autorise à effectuer des mesures à n'importe quel moment d'un algorithme si besoin est.

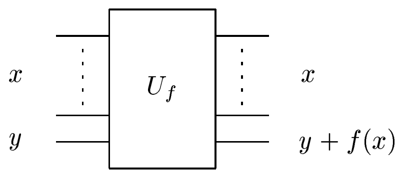
1.8 Sous-procédures quantiques

Le problème posé par une procédure quantique, sous forme donc d'un circuit, est qu'elle renvoie la plupart du temps une superposition de différents états; on effectue alors une mesure qui a une certaine probabilité de renvoyer le bon résultat. Mais si on utilise cette procédure dans un algorithme plus complexe, il faut se souvenir qu'elle ne renvoie non pas la solution, mais bien une superposition, ce qui complique la conception de l'algorithme. Comment remédier à ceci ?

Le paragraphe précédent nous indique une solution possible : dans la description de l'algorithme, on effectue la mesure à la fin de la procédure sans se poser de questions. Si la probabilité qu'elle renvoie la bonne solution est connue, il est alors possible de calculer la probabilité de succès de l'algorithme utilisant cette procédure. Enfin, grâce au résultat que nous venons d'énoncer, on sait qu'il existe un algorithme polynômialement équivalent qui n'effectue qu'une seule mesure à la fin.

2 Algorithmes de recherche

À partir de maintenant, on se donne un oracle f, c'est-à-dire une fonction inconnue dont on peut juste demander l'évaluation en un point ou même en une superposition d'états. Dans les cas qui vont nous occuper, cette fonction f sera à valeurs dans $\{0,1\}$. Ainsi, pour l'évaluer, on utilisera la porte U_f suivante qui agit sur $n+1$ bits, où n est la taille de l'entrée en nombre de bits :



On peut également utiliser la porte S_f qui agit sur n bits seulement et qui inverse la phase de l'entrée x si $f(x)=1$ et laisse l'entrée inchangée sinon. Notons que S_f peut être obtenue à partir de U_f en utilisant un bit interne dans l'état $|b\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ comme $(n+1)^{eme}$ bit d'entrée de U_f. En effet, si on considère un état d'entrée $|\psi\rangle = \sum_{x=0}^{N-1} a_x |x\rangle$, et les ensembles $X_0 = \{x \in X | f(x) = 0\}$ et $X_1 = \{x \in X | f(x) = 1\}$, alors on a :

$$U_f(|\psi\rangle|b\rangle) = \frac{1}{\sqrt{2}} U_f\Big(\sum_{x \in X_0} \big(|x\rangle|0\rangle - |x\rangle|1\rangle\big) + \sum_{x \in X_1} \big(|x\rangle|0\rangle - |x\rangle|1\rangle\big)\Big)$$

$$U_f(|\psi\rangle|b\rangle) = \frac{1}{\sqrt{2}} \Big(\sum_{x \in X_0} \big(|x\rangle|0\rangle - |x\rangle|1\rangle\big) + \sum_{x \in X_1} \big(|x\rangle|1\rangle - |x\rangle|0\rangle\big)\Big)$$

$$U_f(|\psi\rangle|b\rangle) = \Big(\sum_{x \in X_0} |x\rangle - \sum_{x \in X_1} |x\rangle\Big)\Big(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big) = S_f(|\psi\rangle)|b\rangle$$

Le problème de recherche consiste alors à trouver x tel que $f(x)=1$. De manière classique, ce problème se résout en $\Theta(N)$, où N est le cardinal de l'ensemble de départ de f (on a donc $N = 2^n$) : il s'agit d'évaluer f en chacun des points de son ensemble de départ. Le modèle probabiliste classique ne permet pas d'améliorer ce résultat. Nous allons voir maintenant qu'un ordinateur quantique serait plus efficace.

2.1 Algorithme de Grover

L'algorithme de recherche de Grover (1996)[9] est probabiliste et est de complexité $O(\sqrt{N})$, ce qui explique son intérêt par rapport à un algorithme classique. De plus, il est optimal à une constante multiplicative près. Son idée de base est de prendre une superposition de toutes les entrées possibles de f et d'amplifier les amplitudes des solutions.

2.1.1 Inversion par rapport à la moyenne

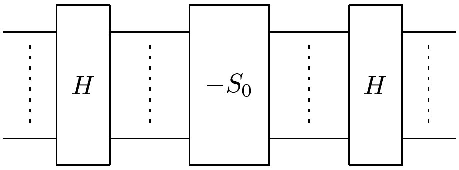
Au cours de l'algorithme, à part la porte S_f, on utilise principalement un circuit quantique M qui effectue une inversion par rapport à la moyenne des amplitudes de tous les états de la base de mesure. Plus précisèment, notons $\sum_{i=0}^{N-1} a_i|i\rangle$ la superposition d'états sur laquelle agit M et A la moyenne des a_i. Lorsque M agit sur $\sum_{i=0}^{N-1} a_i|i\rangle$, on obtient la superposition $\sum_{i=0}^{N-1}(2A - a_i)|i\rangle$.

Pour obtenir cette porte, on utilise la porte de Walsh-Hadamard H et la porte de Dirac en 0. La porte de Dirac est notée S_0 et est définie de la manière suivante :

$$S_0(|0\rangle) = -|0\rangle$$

$$S_0(|i\rangle) = |i\rangle \text{pour i} \neq 0$$

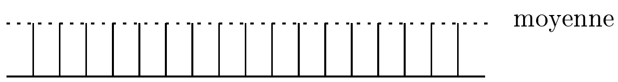
Elle inverse tout simplement la phase de $|0\rangle$ tout en laissant les autres états inchangés. En fait, si note δ_0 la fonction de Dirac en 0, on a $S_0 = S_{\delta_0}$. On montre alors par un calcul que $M = -HS_0H$. On a alors le shéma suivant:

M: 

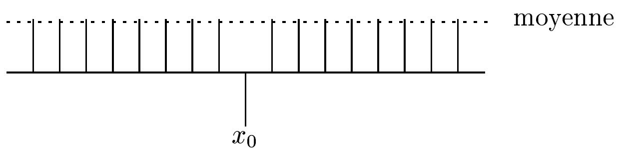
2.1.2 Boucle principale de l'algorithme

Nous allons maintenant décrire le fonctionnement de l'algorithme :

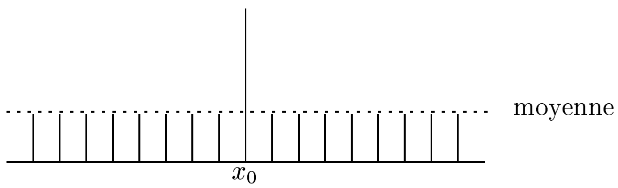
- à partir de l'état $|0\rangle$, et à l'aide de la porte de Walsh-Hadamard, on construit une superposition uniforme de toutes les valeurs possibles de l'entrée x : $\frac{\sum_{i=0}^{N-1}|i\rangle}{\sqrt{N}}$. L'état des n qbits à ce moment peut être décrit par la figure suivante (chaque barre représente l'amplitude d'un i pour $i \in \{0, \ldots, N-1\}$) :



- on applique S_f, c'est-à-dire que l'on inverse la phase des x tels que $f(x) = 1$. En supposant qu'il n'y a qu'une seule solution, la figure précédente a subi la modification suivante :



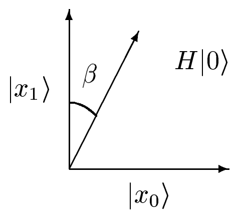
- on effectue à présent une inversion par rapport à la moyenne; celle-ci étant positive, cela amplifie l'amplitude des solutions et diminue l'amplitude des autres. On obtient ainsi le shéma d'amplitudes suivant :



Ainsi arrive-t-on à amplifier l'amplitude des solutions. En répétant cette opération $O(\sqrt{N})$ fois puis en effectuant une mesure, on obtient une solution avec une probabilité d'erreur bornée. En effet, l'amplitude de départ de la solution est $1/\sqrt{N}$, et on cherche à amener cette amplitude à 1. A chaque étape, on augmente cette amplitude d'environ $2A$. Or, au départ, la moyenne A est de $1/\sqrt{N}$. En supposant très grossièrement que la moyenne A reste à peu près constante au cours de l'agorithme, il nous faut donc bien $\frac{1}{2/\sqrt{N}} = O(\sqrt{N})$ itérations pour aboutir au résultat désiré. Ceci n'est bien sûr qu'une explication intuitive de la complexité de l'algorithme. Nous allons voir maintenant comment obtenir rigoureusement ce résultat.

2.1.3 Interprétation géométrique

Pour mieux comprendre comment fonctionne l'algorithme de Grover (notamment l'origine de la complexité en $O(\sqrt{N})$), il est possible de donner une interprétation géométrique de son fonctionnement (Josza 1999)[6] dans le cas où il n'y a qu'une seule solution, notée x_0. Comme nous venons de le voir, l'algorithme de Grover consiste à appliquer un certain nombre de fois la porte suivante : $S_f M = -S_f H S_0 H$. On se place dans le sous-espace vectoriel F engendré par $|x_0\rangle$ et par $H|0\rangle = \frac{\sum_{i=0}^{N-1}|i\rangle}{\sqrt{N}}$ (il s'agit de la superposition uniforme de tous les états de la base de mesure; c'est aussi l'état d'où l'on part dans l'algorithme de Grover). Soit $|x_1\rangle$ orthogonal à $|x_0\rangle$ dans F tel que $(|x_0\rangle, |x_1\rangle)$ forme une base de F. On note β l'angle entre $|x_1\rangle$ et $H|0\rangle$.



Alors on peut interpréter S_f comme une réflexion d'axe $|x_1\rangle$. Des considérations d'algèbre nous montre que $-HS_0H$ est la réflexion d'axe $H|0\rangle$. La composée de deux réflexions est une rotation d'angle le double de l'angle entre les axes des deux réflexions en question. Par conséquent, la boucle décrite ci-dessus revient à effectuer une rotation d'angle 2β. On voit donc qu'en effectuant cette opération à partir de $H|0\rangle$, on se rapproche de $|x_0\rangle$, la solution recherchée, d'un angle 2β.

Mais il est possible en plus d'obtenir de cette manière une évaluation du nombre optimal de rotations pour effectuer une mesure. En effet, pour N grand,

$$\beta \simeq \sin\beta = \langle x_0|H|0\rangle \quad \text{(produit scalaire de } |x_0\rangle \text{ et de } H|0\rangle).$$

Or $H|0\rangle = \frac{\sum_{i=0}^{N-1}|i\rangle}{\sqrt{N}}$. Donc

$$\beta \simeq \frac{1}{\sqrt{N}}$$

Comme on veut s'approcher le plus près possible de $|x_0\rangle$, et que, pour N grand, l'angle entre $H|O\rangle$ et $|x_0\rangle$ est très proche de $\pi/2$, il faut effectuer la rotation $\frac{\pi/2}{2\beta}$ fois. On obtient donc une complexité de $\frac{\pi}{4}\sqrt{N}$.

2.2 Généralisations

Il existe de nombreux moyens de généraliser ce résultat. Nous allons en présenter ici trois (d'après Brassard, Hoyer, Mosca et Tapp en 2000)[5].

2.2.1 Algorithme exact non uniforme

Tout d'abord, il est manifeste que l'algorithme obtenu précédemment est en général inexact. Remarquons cependant que dans le cas où la recherche s'effectue sur seulement deux bits, c'est-à-dire dans un ensemble à 4 éléments, l'algorithme de Grover est exact. Mais dans le cas général, on peut avoir besoin d'un algorithme exact aussi efficace (donc de complexité $O(\sqrt{N})$).

Une des solutions possibles est de remplacer respectivement S_0 et S_f par $S_0(\phi)$ et $S_f(\varphi)$, où ϕ et φ sont deux réels à déterminer par la suite, et $S_0(\phi)$ et $S_f(\varphi)$ sont définis de la manière suivante :

$$S_0(\phi)|i\rangle = \begin{cases} e^{\imath\phi}|i\rangle \text{ si } i = 0 \\ |i\rangle \text{ sinon} \end{cases}$$

$$S_f(\phi)|i\rangle = \begin{cases} e^{\imath\varphi}|i\rangle \text{ si } f(i) = 1 \\ |i\rangle \text{ sinon} \end{cases}$$

Dans le cas de l'algorithme de Grover normal, on effectue une inversion de phase au lieu de multiplier par $e^{\imath\phi}$ ou $e^{\imath\varphi}$. On a donc dans ce cas $\phi = \varphi = \pi$. C'est en ce sens que l'introduction des variables ϕ et φ constitue une généralisation naturelle de l'algorithme de Grover. Sans rentrer plus dans le détail, en choisissant convenablement ϕ et φ, lorsque l'on connaît N ainsi que le nombre de solution(s), il est possible d'obtenir avec certitude une solution au problème de recherche si la solution existe. On remarque cependant que la solution ainsi obtenue n'est pas uniforme, car les réels ϕ et φ dépendent de la taille de l'ensemble de recherche et du nombre de solutions. Un nombre infini de portes quantiques est donc nécessaire pour résoudre le problème de manière globale. Il en est de même pour toutes les autres solutions proposées dans les articles étudiés pendant le stage.

2.2.2 Amplification d'amplitude

Cette autre généralisation est plus intéressante, puisqu'elle permet d'étendre la méthode de Grover à n'importe quel algorithme quantique probabiliste.

On se donne un algorithme quantique A, sous forme de circuit, dont la probabilité de succès est $p > 0$. Alors en itérant la transformation $-AS_0A^{-1}S_f$ $O(1/\sqrt{p})$ fois, on obtient une solution avec grande probabilité. On voit ici également l'avantage par rapport à un algorithme classique qui ne réussit qu'en $O(1/p)$ essais en moyenne.

Dans le cas de l'algorithme de Grover, l'algorithme A consiste juste à appliquer H sur l'état $|0\rangle$ et à effectuer ensuite une mesure sur la superposition uniforme obtenue; par conséquent, s'il n'y a qu'une seule solution, la probabilité de succès est de $1/N$. En appliquant le résultat ci-dessus, on retrouve bien la complexité de $O(\sqrt{N})$ que l'on avait obtenue auparavant.

2.2.3 Estimation du nombre de solutions

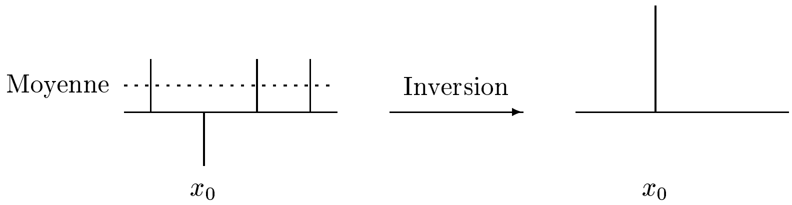
Pour finir avec les généralisations, évoquons rapidement la possibilité de compter le nombre des solutions de l'équation $f(x) = 1$. C'est un cas particulier d'une méthode générale permettant d'estimer la probabilité de succès d'un algorithme quantique.

2.3 Algorithme exact uniforme

Cette partie est un des domaines de recherche proprement dite du stage. Nous allons commencer par préciser quel est exactement le problème que nous allons étudier dans cette section. Nous cherchons ici un algorithme de recherche d'une solution au problème $f(x) = 1$, sachant que f a au plus une solution. Il s'agit donc d'un problème plus restreint que celui présenté précédemment, puisque nous avons ici une condition sur f. Cet algorithme doit vérifier les conditions suivantes : être exact, uniforme et de complexité $O(\sqrt{N})$. A la fin du stage, ce résultat n'a pas été obtenu; en effet, nous n'avons réussi à obtenir qu'un algorithme exact uniforme de complexité $O(N^{\frac{\log 3}{\log 4}})$, ce qui est tout de même mieux qu'un algorithme classique (en effet $\frac{1}{2} < \frac{\log 3}{\log 4} < 1$). Malgré ce résultat semi-négatif, nous allons présenter cet algorithme, ainsi que quelques pistes pour tenter de l'améliorer.

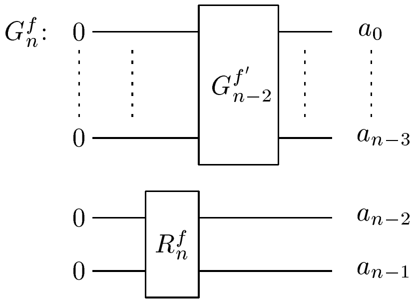
2.3.1 Première présentation de l'algorithme

Nous avons fait remarquer précédemment que l'algorithme de Grover, bien qu'inexact en général, est exact pour $N = 4$ lorsqu'il n'y a qu'une seule solution. En effet, dans ce cas, il suffit d'une application de la boucle de l'algorithme pour résoudre le problème, comme le montre le shéma suivant :



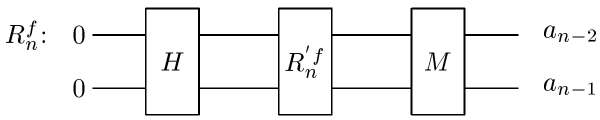
L'idée consiste donc à utiliser cette propriété pour retrouver par récurrence les bits de la solution (si elle existe) deux par deux. Pour ce faire, une approche simple consiste à diviser les tâches pour avoir une vision claire de ce qui se passe. Par la suite, on supposera que le cardinal de l'ensemble de recherche est une puissance de deux ($N = 2^n$). Tout d'abord, soit x_0 l'éventuelle solution de $f(x) = 1$. Soient $a_0, a_1, \ldots, a_{n-1}$ les bits de x_0; on notera tout simplement $x_0 = a_0 a_1 \ldots a_{n-1}$. On définit de manière générale le circuit G_n^f qui, sur l'entrée $|0\rangle$, doit renvoyer x_0 s'il existe; sinon G_n^f renvoie n'importe quoi.

Ensuite, pour appliquer l'idée de récurrence, il est nécessaire d'avoir un circuit R_n^f, qui trouvera les deux derniers bits $a_{n-2}a_{n-1}$ de x_0 à partir de deux qbits initialisés à 0, s'il y a effectivement une solution x_0. Ainsi, on peux déjà exprimer G_n^f en fonction de R_n^f et $G_{n-2}^{f'}$, où f' est la restriction de f au $n-2$ premiers bits, et prend comme deux derniers bits les bits renvoyés par R_n^f, c'est-à-dire $a_{n-2}a_{n-1}$ dans le cas où f a une solution :



Maintenant le problème est d'obtenir R_n^f. Pour ce faire, on introduit le nouveau circuit $R_n^{'f}$ qui décide si les deux bits que ce circuit a en entrée sont $a_{n-2}a_{n-1}$ ou non. De manière plus précise, si f a une solution et que les deux bits en entrée de $R_n^{'f}$ sont les deux derniers bits de la solution, alors $R_n^{'f}$ inverse la phase de ces deux derniers bits et ne fait rien dans le cas contraire.

Si l'on suppose avoir obtenu $R_n^{'f}$, alors il est facile de construire R_n^f; en effet, à ce moment, on peut appliquer l'algorithme exact de Grover sur deux bits :



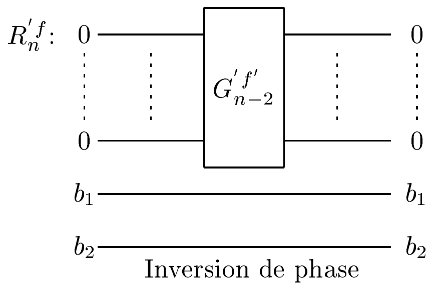
Dans ce shéma, on suppose évidemment qu'il existe une solution; sinon le circuit renvoie n'importe quoi.

A présent, nous allons introduire un dernier circuit, que nous appelerons $G_n^{'f}$, et qui résoud le problème de décision associé à f, c'est-à-dire que, sur l'entrée $|0\rangle$, $G_n^{'f}$ inverse la phase de cette entrée si f admet une solution et ne fait rien sinon. Comment ce circuit peut-il nous aider à construire $R_n^{'f}$? En fait, nous allons faire mieux que cela, puisque pour construire $R_n^{'f}$, nous allons nous servir de $G_{n-2}^{'f'}$, où f' est définie de la même manière que précédemment, ce qui nous évite une définition circulaire de tous ces circuits en fonction des autres et permet de faire fonctionner la récurrence.

Pour réaliser cela, pour construire $R_n^{'f}$ à partir de $G_{n-2}^{'f'}$, on se donne donc deux bits $b_1 b_2$, et on cherche à savoir si $b_1 b_2 = a_{n-2}a_{n-1}$. On considère donc f' obtenu à partir de f, défini sur $n-2$ bits et où les deux derniers bits manquant sont pris comme étant $b_1 b_2$. De deux choses l'une :

- soit f a une solution et $b_1 b_2 = a_{n-2}a_{n-1}$, alors f' a également une solution, à savoir $a_0 \ldots a_{n-3}$;
- soit f n'a pas de solution ou f a une solution mais $b_1 b_2 \neq a_{n-2}a_{n-1}$, alors f' n'a pas de solution.

Donc en appliquant tout simplement $G'^{f'}_{n-2}$ sur $n-2$ bits internes initialisés à 0, on obtiendra une inversion de phase exactement quand f a une solution et que $b_1 b_2 = a_{n-2}a_{n-1}$, ce qui est précisèment l'effet que devait produire R_n^f. On a donc le shéma suivant :



Inversion de phase

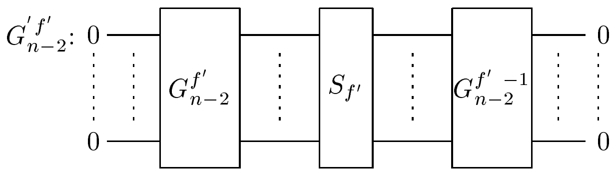
Enfin, pour finir cette présentation, il nous reste à définir $G'^{f'}_{n-2}$ en fonction de $G^{f'}_{n-2}$. Pour ce faire, c'est assez simple; en effet, on applique $G^{f'}_{n-2}$ puis $S_{f'}$. On a alors deux cas :

- f' admet une solution, alors, par hypothèse de récurrence, $G^{f'}_{n-2}$ la trouve. En appliquant $S_{f'}$ à ce moment, on obtient l'inversion de phase recherchée;

- f' n'admet pas de solution; dans ce cas, $G^{f'}_{n-2}$ renvoie n'importe quoi. Comme f' n'a pas de solution, $S_{f'}$ n'a aucun effet, ce qui est bien l'effet voulu.

Mais attention, si l'on s'arrête là, alors l'algorithme obtenu ne fonctionne pas. L'erreur vient du fait que nous avons utilisé des qbits internes lors de la définition de R'^{f}_n. D'après la section 1.4 concernant les qbits internes, ces derniers doivent être remis dans leur état initial, c'est-à-dire 0, après avoir été utilisés. Cela implique donc que $G'^{f'}_{n-2}$ remette son entrée à 0, après avoir éventuellement effectué une inversion de phase. Or, si par exemple f' admet effectivement une solution y_0, l'entrée de $G'^{f'}_{n-2}$ se retrouve dans l'état y_0 à la fin du calcul, et dans le cas contraire on n'a aucune information sur l'état des bits sur lesquels travaille $G'^{f'}_{n-2}$.

Comment remédier à cela ? La solution consiste à revenir simplement sur nos pas et d'appliquer à ce moment $G^{f'\,-1}_{n-2}$. Ainsi, on revient à l'état initial en ayant simplement obtenu l'inversion de phase que l'on recherchait le cas échéant. Enfin, pour définir $G^{f'\,-1}_{n-2}$, il n'y a pas de problème; il suffit de prendre le circuit symétrique de $G^{f'}_{n-2}$. De manière générale, pour obtenir le circuit effectuant la transformation inverse d'un circuit quelconque, il faudrait également que chaque porte de base soit remplacée par la porte inverse. Mais, dans notre cas, toute les transformations de base que nous avons utilisées sont des involutions; par conséquent, nous n'avons pas besoin de chercher comment inverser les portes H et M. On obtient ainsi le shéma suivant (au cours duquel s'effectue une inversion de phase s'il y a une solution) :



On construit ainsi par récurrence G_n^f. Pour démarrer la récurrence, comme on exprime G_n^f en fonction de $G^{f'}_{n-2}$, il faut distinguer les cas n pair et n impair :

- si n est pair, alors la récurrence démarre avec G_2; or on sait résoudre le problème de manière exacte lorsqu'il n'y a que deux qbits;

- si n est impair, on ajoute un qbit supplémentaire à l'ensemble de recherche et on se ramène au cas précédent.

On montre par récurrence que l'algorithme que nous venons de décrire est exact.

2.3.2 Complexité de l'algorithme

Notons respectivement $Q(A)$ et $C(A)$ la complexité en nombre de requêtes et en nombre de portes d'un circuit quelconque A. Alors (on ne note plus la fonction en exposant, car la complexité ne dépend que du nombre de bits n) :

$$\begin{aligned}
Q(G_n) &= Q(R_n) + Q(G_{n-2}) \\
&= Q(R'_n) + Q(G_{n-2}) \\
&= Q(G'_{n-2}) + Q(G_{n-2}) \\
&= (2Q(G_{n-2}) + 1) + Q(G_{n-2}) \\
Q(G_n) &= 3Q(G_{n-2}) + 1
\end{aligned}$$

Et pour la complexité en nombre de portes :

$$\begin{aligned}
C(G_n) &= C(R_n) + C(G_{n-2}) \\
&= (C(R'_n) + 7) + C(G_{n-2}) \\
&= (C(G'_{n-2}) + 7) + C(G_{n-2}) \\
&= ((2C(G_{n-2}) + 1) + 7) + C(G_{n-2}) \\
C(G_n) &= 3C(G_{n-2}) + 8
\end{aligned}$$

Pour évaluer la complexité en nombre de portes de R_n, il faut se rappeler que la transformation de Walsh-Hadamard est obtenue en appliquant la porte d'Hadamard sur chaque qbit, et que l'inversion par rapport à la moyenne M se construit à partir de deux transformations de Walsh-Hadamard et de la porte de Dirac en 0. Comme tout ceci se passe sur 2 qbits, on arrive bien à un total de 7 portes. On a de plus les coûts suivants pour $n = 2$: $Q(G_2) = 1$ et $C(G_2) = 8$. Ces deux récurrences se résolvent respectivement en :

$$Q(G_{2n}) = \frac{3^n - 1}{2}$$

et

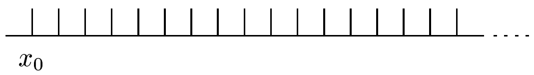
$$C(G_{2n}) = 4(3^n - 1)$$

ce qui donne dans les deux cas une complexité en $O(3^{n/2})$. Or $n = \log_2 N$, donc la complexité est de $O(e^{\frac{\log_2 N * \log 3}{2}}) = O(N^{\frac{\log 3}{\log 4}})$, ce qui est bien le résultat annoncé en début de section.

2.3.3 Deuxième présentation de l'algorithme

Dans cette présentation, on utilise uniquement des shémas d'amplitudes. On procède là aussi par récurrence et on suppose que le nombre de qbits est pair; ceci implique que le cardinal de l'ensemble de recherche est une puissance de quatre. On a également besoin d'effectuer une inversion par rapport à la moyenne non plus globale, mais par ensembles dont le cardinal est une puissance de quatre. Cela revient en fait à faire une inversion par rapport à la moyenne sur un nombre limité de bits. Par exemple, par obtenir une inversion par rapport à la moyenne sur des groupes de quatre éléments, il suffit de la restreindre à deux bits.

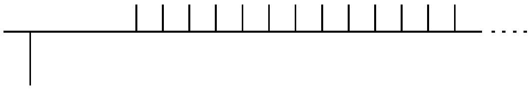
On cherche donc la solution de $f(x) = 1$ et les seules opérations autorisées sont l'inversion de phase de l'éventuelle solution et l'inversion par rapport à la moyenne par groupes dont le cardinal est une puissance de quatre. On part de la superposition uniforme :



Par commodité, on place l'éventuelle solution en première position. Ensuite, on suppose par récurrence que l'on a réussi à remettre à zéro les 4^k premières amplitudes à part la première, c'est-à-dire celle de la solution, et que cette opération n'a à aucun moment modifié les amplitudes situées en-dehors de ce groupe de 4^k (dans les schémas, on prend le cas $k = 1$, mais ce qui se passe pour k quelconque est très similaire) :

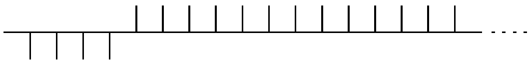
x_0

Alors, on inverse la phase de x_0 :



x_0

Ensuite, on effectue à l'envers toutes les étapes qui nous avaient permis d'arriver au deuxième schéma; par hypothèse de récurrence, ces étapes n'ont pas influencé d'amplitudes en-dehors des 4^k premières, et la phase globale des 4^k premières amplitudes a été inversée, donc on se retrouve dans la situation suivante :

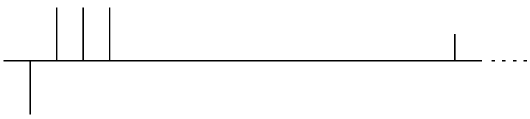


x_0

A ce moment, la moyenne sur les 4^{k+1} premières amplitudes est de $1/2$; en effectuant une inversion par rapport à la moyenne par groupes de 4^{k+1}, on obtient la situation suivante :

x_0

On inverse alors de nouveau la phase de x_0 :



x_0

Pour finir, on effectue encore une inversion par rapport à la moyenne par groupes de 4^k :

x_0

Toutes ces opérations n'ont affecté que les 4^{k+1} premières amplitudes, donc les conditions imposées par la récurrence sont bien réunies. Enfin, on remarque que la superposition uniforme de départ vérifie les conditions exigées pour $k = 0$, ce qui permet d'amorcer la récurrence.

Ainsi, nous voyons plus clairement ce que fait notre algorithme en pratique; en fait, l'agorithme décrit dans la première présentation ajoute une étape supplémentaire qui permet d'effectuer uniquement des inversions par rapport à la moyenne sur deux bits, c'est-à-dire par groupes de quatre, mais le principe de fonctionnement reste le même.

2.3.4 Récurrence sur k qbits

Dans cette section et la suivante, nous présentons deux idées pour tenter d'améliorer ce résultat. Tout d'abord, on peut penser à généraliser la méthode précédente, pour trouver les bits non plus deux par deux mais de manière générale k par k, où k peut éventuellement dépendre de n. Pour ce faire, on découpe le problème d'une façon similaire :

- $G_n^k(S_f)$ trouve les k derniers bits de la solution associée à S_f;
- $G_n(S_f) = G_n^n(S_f)$ trouve donc la solution;
- $S_n^k(S_f)$ décide si les k bits en entrée sont les k derniers bits de la solution;
- $G'_n(S_f)$ décide s'il existe une solution.

Dans le cas où f n'a pas de solution, ce que renvoient les circuits définis ci-dessus importe peu. On peut alors les définir récursivement :

- pour commencer, $G_n(S_f)$ s'obtient de manière évidente en appliquant successivement $G_n^k(S_f)$ puis $G_{n-k}(S_{f'})$, où f' est définie à partir de S_f en prenant comme k derniers bits les bits renvoyés par $G_n^k(S_f)$.
- ensuite, on remarque que $G_n^k(S_f)$ est exactement égal à $G_k(S_n^k(S_f))$; en effet, $G_k(S_n^k(S_f))$ trouve la solution de son argument $S_n^k(S_f)$, c'est-à-dire l'entrée dont $S_n^k(S_f)$ va inverser la phase. Or par définition de $S_n^k(S_f)$, ce dernier n'inverse la phase de son entrée uniquement dans le cas où son entrée est égale aux k derniers bits de la solution de S_f. Donc $G_k(S_n^k(S_f))$ va trouver les k derniers bits de l'éventuelle solution de S_f, ce qui est bien ce que l'on attend de $G_n^k(S_f)$;
- pour finir, $S_n^k(S_f)$ et $G'_n(S_f)$ s'obtiennent respectivement à partir de $G'_{n-k}(S_f)$ et de $G_n(S_f)$ de la même manière que dans la première version de l'algorithme.

Une rapide analyse de complexité nous mène à l'équation suivante :

$$C(G_n) = 2C(G_k)C(G_{n-k}) + C(G_k) + C(G_{n-k})$$

Pour $k = 2$, on retrouve l'équation précédente (car $C(G_k) = 1$). Une autre valeur naturelle que l'on peut essayer pour k est $n/2$, pour tenter de procéder par dichotomie. A une approximation près (ou en supposant que n est une puissance de 2), on a alors :

$$C(G_n) = 2(C(G_{n/2})^2 + C(G_{n/2}))$$

On montre alors facilement que $C(G_n) \geq 2^n$, ce qui donne une complexité au moins linéaire; par conséquent, il ne s'agit pas d'une bonne solution.

2.3.5 Problème de décision pour $N = 4$

Une autre idée consiste à essayer de trouver une autre solution pour exprimer G'_n en fonction de G_n; il est clair par exemple que si on arrive à construire G'_n en utilisant une seule fois G_n, la complexité globale de l'algorithme en est grandement affectée. Plus précisèment, on aurait alors l'équation suivante :

$$C(G_n) = 2C(G_{n-2}) + 1$$

qui se résout facilement en $C(G_{2n}) = 2^n - 1$, ce qui donnerait une complexité en $O(\sqrt{N})$, qui est le résultat recherché.

Plus simplement, on peut se demander combien d'appels à f sont nécessaires pour décider si f a une solution dans le cas où $N = 4$ (on se situe toujours dans le cadre où f a au plus une solution). On sait déjà que trois appels suffisent : on trouve la solution avec deux bits internes initialisés à zéro en un appel, on vérifie si le résultat obtenu est bien la solution en un deuxième appel à f, enfin on remet les deux bits internes utilisés à zéro à l'aide d'un troisième et dernier appel à f. Est-il possible d'effectuer la même opération avec seulement deux voire un seul appel à f ?

En fait, on peut montrer par un calcul qu'un seul appel à f ne suffit pas. Ceci se montre par l'absurde. On suppose donc qu'il existe un algorithme quantique qui, sur l'entrée $|0\rangle$ renvoie $|1\rangle$ s'il y a une solution au problème $f(x) = 1$ dans le cas où $N = 4$, et $|0\rangle$ sinon. On suppose de plus que cet algorithme ne fait qu'un seul appel à f, cet appel s'effectuant par l'intermédiaire de la porte U_f. On se place alors juste avant cet appel à U_f. Cet appel s'effectue sur un 3-qbit; notons $|b\rangle$ l'état global du système au moment de l'appel. On peut alors décomposer $|b\rangle$ de la manière suivante :

$$|b\rangle = \beta_0|0\rangle|b_0\rangle + \ldots + \beta_7|7\rangle|b_7\rangle$$

Dans cette décomposition, la première composante de chaque terme correspond à un état possible du 3-qbits sur lequel va être appliqué U_f, et la deuxième composante est l'état du reste du système auquel cet état particulier du 3-qbits est corrélé. Les β sont alors les amplitudes associées.

On applique donc U_f sur les 3 premiers qbits de $|b\rangle$. Si l'équation $f(x) = 1$ n'a pas de solution, alors $|b\rangle$ reste inchangé. Sinon, on note comme d'habitude x_0 la solution de f, et $|c_{x_0}\rangle$ l'état dans lequel se retrouve le système après application de U_f. Comme cette dernière inverse le troisième qbit de $|b\rangle$ si les deux premiers sont la solution, on a alors :

$$|c_{x_0}\rangle = |b\rangle - \beta_{2x_0}|2x_0\rangle|b_{2x_0}\rangle - \beta_{2x_0+1}|2x_0+1\rangle|b_{2x_0+1}\rangle + \beta_{2x_0}|2x_0+1\rangle|b_{2x_0}\rangle + \beta_{2x_0+1}|2x_0\rangle|b_{2x_0+1}\rangle$$

Par la suite, l'algorithme ne fait plus appel à U_f, par conséquent $|b\rangle$, $|c_0\rangle$,...,$|c_3\rangle$ subissent les même transformations. On a donc les évolutions suivantes :

$$|b\rangle \rightarrow |0\rangle|\psi\rangle$$

$$|c_{x_0}\rangle \rightarrow |1\rangle|\psi\rangle$$

Ici, $|\psi\rangle$ représente l'état des bits internes. Or, ces derniers ont nécessairement été remis à zéro avant la fin de l'algorithme, donc $|\psi\rangle = |0\rangle$. On remarque alors que $\langle\psi, 1|0, \psi\rangle = \langle\psi|\psi\rangle\langle 1|0\rangle = 0$. Par conséquent, comme les transformations utilisées dans l'algorithme sont unitaires, elles conservent le produit scalaire et on a $\langle c_{x_0}|b\rangle = 0$ pour tout x_0. De plus, $|b\rangle$ est normé donc $\langle b|b\rangle = 1$. Donc :

$$\langle c_{x_0}|b\rangle = \langle b|b\rangle - |\beta_{2x_0}|^2\langle b_{2x_0}|b_{2x_0}\rangle - |\beta_{2x_0+1}|^2\langle b_{2x_0+1}|b_{2x_0+1}\rangle + \overline{\beta_{2x_0}}\beta_{2x_0+1}\langle b_{2x_0}|b_{2x_0+1}\rangle + \overline{\beta_{2x_0+1}}\beta_{2x_0}\langle b_{2x_0+1}|b_{2x_0}\rangle$$

$$\langle c_{x_0}|b\rangle = 1 - ||\beta_{2x_0}|b_{2x_0}\rangle - \beta_{2x_0+1}|b_{2x_0+1}\rangle||^2 = 0$$

$$||\beta_{2x_0}|b_{2x_0}\rangle - \beta_{2x_0+1}|b_{2x_0+1}\rangle||^2 = 1$$

Par inégalité triangulaire :

$$|\beta_{2x_0}| \; || \; |b_{2x_0}\rangle|| + |\beta_{2x_0+1}| \; || \; |b_{2x_0+1}\rangle|| \geq 1$$

Posons $u = |\beta_{2x_0}| \; || \; |b_{2x_0}\rangle||$ et $v = |\beta_{2x_0+1}| \; || \; |b_{2x_0+1}\rangle||$. Alors $u + v \geq 1$. Deux cas se présente alors :

- si $u \geq 1$ alors $u^2 \geq 1$ donc $u^2 + v^2 \geq 1$;
- si $u \leq 1$, alors $v \geq 1 - u \geq 0$ donc $v^2 \geq (1-u)^2$ et $u^2 + v^2 \geq u^2 + (1-u)^2 \geq 0,5$.

Dans tous les cas, on $u^2 + v^2 \geq 0,5$. On obtient donc l'inégalité suivante :

$$|\beta_{2x_0}|^2|| \; |b_{2x_0}\rangle||^2 + |\beta_{2x_0+1}|^2|| \; |b_{2x_0+1}\rangle||^2 \geq 0,5$$

Or, cette inégalité est vraie pour tout x_0, car notre algorithme doit fonctionner dans tous les cas. Donc, on obtient les 4 inégalités suivantes :

$$|\beta_0|^2|| \; |b_0\rangle||^2 + |\beta_1|^2|| \; |b_1\rangle||^2 \geq 0,5$$

$$|\beta_2|^2|| \; |b_2\rangle||^2 + |\beta_3|^2|| \; |b_3\rangle||^2 \geq 0,5$$

$$|\beta_4|^2||\ |b_4\rangle||^2 + |\beta_5|^2||\ |b_5\rangle||^2 \geq 0,5$$
$$|\beta_6|^2||\ |b_6\rangle||^2 + |\beta_7|^2||\ |b_7\rangle||^2 \geq 0,5$$

Il suffit maintenant de calculer $\langle b|b\rangle$ pour obtenir une contradiction :

$$1 = \langle b|b\rangle = |\beta_0|^2||\ |b_0\rangle||^2 + \ldots + |\beta_7|^2||\ |b_7\rangle||^2 \geq 4 * 0,5 = 2$$

Par conséquent, on obtient une borne inférieure de deux appels à f pour résoudre le problème de décision de l'existence d'une solution à l'équation $f(x) = 1$. Le problème, qui reste ouvert, consiste donc à savoir si ce nombre minimum d'appel à f est deux ou trois.

3 Problème des collisions

3.1 Énoncé du problème

On se donne une fonction $f : X \to Y$ (X et Y ensembles finis) . On cherche alors deux éléments distincts x_1 et x_2 de X tels que $f(x_1) = f(x_2)$. On dit alors que le couple (x_1, x_2) est une collision simple. De manière générale, on dit que f admet une k-collision si et seulement s'il existe $x_1, \ldots, x_k$ k éléments distincts de X tels que $f(x_1) = \ldots = f(x_k)$.

On suppose sans perte de généralité que $X = \{1, \ldots, N\}$ et $Y = \{1, \ldots, M\}$. Alors de manière classique, pour trouver une collision de f (simple ou multiple), il suffit de trier $f(1), \ldots, f(N)$, puis de parcourir la liste ainsi obtenue en comparant des éléments consécutifs. Toute cette opération s'effectue donc en $O(N \log N)$.

Par conséquent, l'objectif est d'obtenir en utilisant un algorithme quantique une complexité moindre pour résoudre ce problème. Burhman, Dürr, Heiligman, Høyer, Magniez, Santha et de Wolf ont exhibé une borne supérieure de $O(N^{3/4} \log N)$ pour les collisions simples et de $O(N^{1-\frac{1}{2^k}} \log N)$ pour les k-collisions de manière générale [2], ainsi qu'une borne inférieure de $\Omega(\sqrt{N})$ pour tous les types de collision.

L'objectif de cette section était de mettre en forme une autre manière plus directe d'obtenir ces résultats.

3.2 Collisions simples

Pour commencer, soit A une partie quelconque de X. On définit alors la fonction g_A de X dans $\{0, 1\}$ de la manière suivante, pour $x \in X$: $g_A(x) = 1$ s'il existe $a \in A$ tel que (x, a) forme une collision, c'est-à-dire tel que $f(x) = f(a)$ et $x \neq a$; $g_A(x) = 0$ sinon.

Ensuite on considère la fonction F de l'ensemble des parties de X dans $\{0, 1\}$ définie par : $F(A) = 1$ s'il existe $x \in X$ tel que $g_A(x) = 1$; $F(A) = 0$ sinon.

L'algorithme se déroule ensuite en plusieurs étapes :

1. tout d'abord, on crée une partition de X en N/k sous-ensembles de cardinal k : $A_1, \ldots, A_{N/k}$;
2. ensuite, en utilisant l'algorithme de Grover, on cherche $i \in \{1, \ldots, N/k\}$ tel que $F(A_i) = 1$;
3. $F(A_i) = 1$ donc il existe $x \in X$ tel que $g_A(x) = 1$; on cherche donc cet x avec l'algorithme de Grover;
4. enfin, $g_A(x) = 1$, donc il existe $a \in A_i$ tel que (x, a) forme une collision simple; en utilisant une dernière fois l'algorithme de Grover, on cherche cet a et on obtient la collision recherchée.

Ceci n'est bien sûr que le squelette de l'algorithme, puisqu'il manque le calcul des fonctions F et g_A. Remarquons également que l'on utilise de nombreuses fois l'algorithme de Grover comme procédure; cela ne pose pas de problèmes d'après la section 1.8 sur les sous-procédures quantiques.

Reprenons donc notre algorithme en analysant la complexité à chaque étape (on note C la complexité en nombre de portes et Q la complexité en nombre de requêtes) tout en précisant comment s'effectuent les calculs :

1. création d'une partition de X : $C = O(N/k)$ et $Q = 0$;
2. - pour trouver A_i, d'après l'analyse de la complexité de l'algorithme de Grover, $C = O(\sqrt{N/k}\, C(F))$ et $Q = O(\sqrt{N/k}\, Q(F))$;

- pour calculer $F(A)$, on cherche avec l'algorithme de Grover un $x \in X$ tel que $g_A(x) = 1$, donc $C(F) = \sqrt{N}\ C(g_A)$ et $Q(F) = \sqrt{N}\ Q(g_A)$;
- enfin, pour calculer $g_A(x)$, on utilise une nouvelle fois l'algorithme de Grover pour chercher $a \in A$ tel que $f(a) = f(x)$, ce qui coûte $O(\sqrt{k})$;
- on obtient donc la complexité :

$$C = Q = O(\sqrt{N/k}\sqrt{N}\sqrt{k}) = O(N)$$

3. nous venons de voir le coût du calcul de $g_A(x)$, on en déduit la complexité de cette troisième étape : $C = Q = O(\sqrt{N}\sqrt{k})$;
4. cette dernière étape est juste une application de l'algorithme de Grover, donc la complexité est simple à calculer : $C = Q = O(\sqrt{k})$.

Par conséquent, on obtient la complexité globale suivante :

$$C = O(N/k + N + \sqrt{Nk} + \sqrt{k}) \text{ et } Q = O(N + \sqrt{Nk} + \sqrt{k})$$

Comme on recherchait une complexité en $O(N^{3/4}\log N)$, on s'aperçoit que l'algorithme que nous venons de décrire n'est pas suffisamment efficace, à cause du terme en $O(N)$. Comment y remédier ?

La solution consiste à accélerer le calcul de $g_A(x)$ en effectuant des calculs préliminaires. Ainsi, la deuxième étape va se dérouler plutôt de la manière suivante :

- on cherche A_i comme précédemment à l'aide de l'algorithme de Grover;
- pour calculer $F(A)$, par contre, on commence par calculer $f(a)$ pour tous les $a \in A$; on conserve le résultat de ce calcul sous forme de couples $(a, f(a))$ pour tout $a \in A$. Puis on effectue un tri suivant les valeurs en f des éléments de A. Toute cette opération coûte $C = O(k + k\log k) = O(k\log k)$ et $Q = O(k)$. Ensuite, on utilise l'algorithme de Grover de la même manière qu'auparavant;
- pour calculer $g_A(x)$, on commence par calculer $f(x)$, puis on utilise non plus l'algorithme de Grover, mais plutôt une recherche classique dans l'ensemble ordonné des $(a, f(a))_{a\in A}$, ce qui coûte $C = O(\log k)$ et $Q = O(1)$;

La complexité de la deuxième étape devient donc :

$$C = O(\sqrt{\frac{N}{k}}(k\log k + \sqrt{N}\log k)) = O((\sqrt{Nk} + \frac{N}{\sqrt{k}})\log k)$$

$$Q = O(\sqrt{\frac{N}{k}}(k + \sqrt{N})) = O(\sqrt{Nk} + \frac{N}{\sqrt{k}})$$

On peut utiliser le même procédé dans la troisième étape de l'algorithme, ce qui donne une complexité de $C = O(k\log k + \sqrt{N}\log k)$ et $Q = O(k + \sqrt{N})$.

Pour minimiser la complexité globale de l'algorithme, on prend $k = \sqrt{N}$; on obtient alors les complexités suivantes pour l'ensemble de l'algorithme :

$$C = O(N^{3/4}\log N) \text{ et } Q = O(N^{3/4})$$

On a donc bien ainsi le résultat voulu.

3.3 Analyse de probabilité

Comme on utilise de nombreuses fois l'algorithme de Grover, dont la probabilité d'échec est non nulle, on peut se demander quelle est la probabilité de succès de notre algorithme. Pour ce faire, on se sert du résultat suivant lequel la probabilité d'échec de l'algorithme de Grover sur un ensemble à N éléments est proche de $1/N$ pour N grand, et on procède étape par étape :

1. la première étape ne pose pas de problème;
2.
   - le calcul de $g_A(x)$ n'utilise pas l'algorithme de Grover; il est donc exact;
   - le calcul de $F(A)$ se fait avec une probabilité de succès de $1 - 1/N$;
   - lors de la recherche de A_i, F est calculée $O(\sqrt{\sqrt{N}}) = O(N^{1/4})$ fois; de plus, il faut prendre en compte la probabilité d'échec de l'algorithme de Grover que l'on utilise pour trouver A; la probabilité de succès à ce moment est donc de $(1-1/N)^{KN^{1/4}}(1-1/\sqrt{N})$ (où K est une constante);
3. il s'agit d'une application directe de l'algorithme de Grover; la probabilité de succès est de $1 - 1/N$;
4. de manière similaire à l'étape précédente, la probabilité de succès est de $1 - 1/\sqrt{N}$;

On obtient donc la probabilité de succès suivante :

$$(1 - 1/N)^{KN^{1/4}}(1 - 1/N)(1 - 1/\sqrt{N})^2$$

Or $(1 - 1/N)^{KN^{1/4}} \sim e^{-KN^{-3/4}}$. Donc la probabilité de succès tend vers 1 quand N tend vers l'infini. Ceci montre donc la validité de l'algorithme.

3.4 k-collisions

Nous allons expliquer dans cette section comment fonctionne l'algorithme pour les 3-collisions. Ceci nous permet de voir comment adapter l'algorithme pour les collisions simples que nous venons de voir pour les k-collisions. Comme précédemment, on commence par définir quelques fonctions :

- la fonction $g_{A,B}$ (où A et B sont deux parties de X) de X dans $\{0, 1\}$ définie par : pour $x \in X$, $g_{A,B}(x) = 1$ s'il existe $a \in A$ et $b \in B$ tel que (x, a, b) forme une 3-collision et $g_A(x, y) = 0$ sinon;
- la fonction F de $\mathcal{P}(X)$ dans $\{0, 1\}$ définie par : pour $A \in \mathcal{P}(X)$, $F(A) = 1$ s'il existe $a \in A$, et $x, y \in X$ tels que (x, y, a) forme une 3-collision et $F(A) = 0$ sinon;
- la fonction G de $\mathcal{P}(X)^2$ dans $\{0, 1\}$ définie par : pour $A, B \in \mathcal{P}(X)$, $G(A, B) = 1$ s'il existe $a \in A$, $b \in B$ et $x \in X$ tels que (x, a, b) forme une 3-collision et $G(A, B) = 0$ sinon.

On effectue alors les opérations suivantes (on calcule la complexité simultanèment) :

1. on crée une partition de X en N/k sous-ensembles de cardinal k : $A_1, \ldots, A_{N/k}$. Le coût est de $C = O(N/k)$ et $Q = 0$;
2. on crée une deuxième partition de X en N/l sous-ensembles de cardinal l : $B_1, \ldots, B_{N/l}$. On a alors de même qu'à l'étape précédente : $C = O(N/l)$ et $Q = 0$;
3. on cherche avec l'algorithme de Grover $i \in \{1, \ldots, N/k\}$ tel que $F(A_i) = 1$: $C = O(\sqrt{N/k}\ C(F))$ et $Q = (\sqrt{N/k}\ Q(F))$. Pour calculer $F(A)$:
   - on commence par calculer $f(a)$ pour tous les $a \in A$. Ensuite, on effectue un tri suivant les valeurs en f des éléments de A. Tout ceci coûte $C = 0(k \log k)$ et $Q = O(k)$.
   - on cherche ensuite avec l'algorithme de Grover un $j \in \{1, \ldots, N/l\}$ tel que $G(A, B_j) = 1$. Pour calculer $G(A, B)$:
     - on commence par calculer $f(b)$ pour tous les $b \in B$. Comme d'habitude, on trie les éléments de B suivant les valeurs en f, ce qui coûte $C = O(l \log l)$ et $Q = O(l)$;
     - on cherche ensuite un $x \in X$ tel que $g_{A,B}(x) = 1$ avec l'algorithme de Grover : $C = 0(\sqrt{N}C(g_{A,B}))$ et $Q = (\sqrt{N}Q(g_{A,B}))$;
     - enfin, pour calculer $g_{A,B}(x)$, on remarque qu'à cet endroit de l'algorithme on a déjà calculé et trié les valeurs en f des ensembles A et B; il suffit donc de faire une recherche dichotomique dans chacun des ensembles, ce qui coûte $C = O(\log k + \log l)$ et $Q = O(1)$.

Le coût global du calcul de $G(A,B)$ est donc de $C = O\big(l\log l + \sqrt{N}(\log k + \log l)\big)$ et $Q = O(l+\sqrt{N})$.

On en déduit immédiatement le coût de calcul de cette étape :

$$C = O\left(\sqrt{\frac{N}{k}}\left(k\log k + \sqrt{\frac{N}{l}}(l\log l + \sqrt{N}(\log k + \log l))\right)\right) \text{ et } Q = O\left(\sqrt{\frac{N}{k}}\left(k + \sqrt{\frac{N}{l}}(l+\sqrt{N})\right)\right)$$

4. une fois que l'on a trouvé A, on calcule les valeurs en f de tous ses éléments puis on les trie suivant ces valeurs; ceci coûte $C = O(k\log k)$ et $Q = O(k)$;
5. on cherche maintenant un $j \in \{1,\ldots,N/l\}$tel que $G(A,B_j)$ de la même manière que dans la deuxième étape, ce qui coûte donc :

$$C = O\left(\sqrt{\frac{N}{l}}(l\log l + \sqrt{N}(\log k + \log l))\right)) \text{ et } Q = O\left(\sqrt{\frac{N}{l}}(l+\sqrt{N})\right)$$

6. de même qu'à l'étape 4, on calcule les valeurs en f des éléments de B; on les trie ensuite suivant ces valeurs : $C = O(l\log l)$ et $Q = O(l)$;
7. maintenant que l'on connaît A et B, il suffit de chercher $x \in X$ tel que $g_{A,B}(x) = 1$. La complexité de cette opération a déjà été calculée précédemment : $C = O(\sqrt{N}(\log k + \log l))$ et $Q = (\sqrt{N})$;
8. il reste pour finir à chercher $a \in A$ et $b \in B$ tels que (x,a,b) forme une 3-collision (recherche classique dans des ensembles ordonnés): $C = O(\log k + \log l)$ et $Q = O(1)$.

Il suffit maintenant d'ajouter la complexité de chaque étape pour obtenir la complexité globale de l'algorithme :

$$C = O\left(\frac{N}{k} + \frac{N}{l} + \sqrt{\frac{N}{k}}\left(k\log k + \sqrt{\frac{N}{l}}(l\log l + \sqrt{N}(\log k + \log l))\right)\right)$$

$$Q = O\left(\sqrt{\frac{N}{k}}\left(k + \sqrt{\frac{N}{l}}(l+\sqrt{N})\right)\right)$$

On obtient alors la plus faible complexité an choisissant $l = \sqrt{N}$ et $k = N^{3/4}$. On a alors :

$$C = O(N^{7/8}\log N) \text{ et } Q = O(N^{7/8})$$

On retrouve bien le résultat annoncé en début de section car $7/8 = 1 - 1/2^3$.

La généralisation aux k-collisions se fait ensuite de manière automatique, bien que très lourde à décrire; c'est pourquoi nous nous sommes limité à expliquer le fonctionnement de l'algorithme pour les 3-collisions. Une analyse de probabilité similaire à celle exposée pour les collisions simples montre que la probabilité de succès de cet algorithme est de (avec K et K' sont deux constantes) :

$$\left(1-\frac{1}{N}\right)^{KK'N^{3/8}}\left(1-\frac{1}{N}\right)^{KN^{1/4}}\left(1-\frac{1}{N}\right)\left(1-\frac{1}{\sqrt{N}}\right)^{KN^{1/8}}\left(1-\frac{1}{\sqrt{N}}\right)\left(1-\frac{1}{N^{1/4}}\right)$$

La probabilité de succès tend bien vers 1 lorsque N tend vers l'infini. En effet, on a :

$$\left(1-\frac{1}{N}\right)^{KK'N^{3/8}} \sim e^{-KK'N^{-5/8}}$$

$$\left(1-\frac{1}{N}\right)^{KN^{1/4}} \sim e^{-KN^{-3/4}}$$

$$\left(1-\frac{1}{\sqrt{N}}\right)^{KN^{1/8}} \sim e^{-KN^{-3/8}}$$

Conclusion

Le calcul quantique est un domaine très récent de l'informatique, ce qui explique la présence de nombreux problèmes ouverts. Les méthodes que l'on y emploie sont très différentes de l'informatique classique et il est plus difficile d'y avoir une bonne intuition. Cependant, les résultats obtenus dans ce domaine justifient de s'y intéresser et d'essayer de surmonter les difficultés physiques rencontrées.

Au cours de ce rapport, nous avons présenté tout d'abord un modèle de calcul quantique utilisant la notion de circuits. Ensuite, nous avons expliqué l'algorithme de Grover, évoqué quelques généralisations puis décrit en détails une variante exacte uniforme de cet algorithme. Enfin, nous avons exposé le problème des collisions et un nouvel algorithme quantique pour résoudre ce problème de manière efficace.

La création d'un algorithme de recherche exact uniforme de complexité $O(\sqrt{N})$, ou encore la complexité du problème des collisions en calcul quantique restent des problèmes ouverts.

Remerciements

Pour l'aide précieuse apportée au cours du stage :

Frédéric Magniez,

Miklos Santha,

Sophie Laplante.

Références

[**1**] E.Rieffel et W.Polak, 2000, An introduction to quamtum computing for non-physicists, http://xxx.lpthe.jussieu.fr/archive/quant-ph/9809016 v2

[**2**] H.Buhrman, C.Dürr, M.Heiligman, P.Høyer, F.Magniez, M.Santha et R.de Wolf, 2000, Quantum algorithms for element distinctness, http://xxx.lpthe.jussieu.fr/archive/quant-ph/0007016

[**3**] D.Aharonov, A.Kitaev et N.Nisan, 1998, Quantum circuits with mixed states, http://xxx.lpthe.jussieu.fr/archive/quant-ph/980629

[**4**] C.H.Benett, E.Bernstien, G.Brassard et U.Vazirani, 1997, Strengths and weaknesses of quantum computing, http://xxx.lpthe.jussieu.fr/archive/quant-ph/9701001

[**5**] G.Brassard, P.Høyer, M.Mosca et A.Tapp, 2000, Quantum amplitude amplification and estimation, http://xxx.lpthe.jussieu.fr/archive/quant-ph/000505

[**6**] R.Jozsa, 1999, Searching in Grover's algorithm, http://xxx.lpthe.jussieu.fr/archive/quant-ph/9901021 v1

[**7**] R.P.Feynman, 1982, Simulating physics with computers, *International Journal of Theoretical Physics 21*, p. 467-488

[**8**] P.W.Shor, 1994, Algorithms for quantum computation: Discrete log and factoring, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, p. 124-134

[**9**] L.K.Grover, 1996, A fast quantum mechanical algorithm for database search, http://xxx.lpthe.jussieu.fr/archive/quant-ph/9605043

[**10**] A.Barenco, C.H.Bennett, R.Cleve, D.P.DiVincenzo, N.H.Margolus, P.W.Shor, T.Sleator, J.A.Smolin et H.Weinfurter, 1995, Elementary gates for quantum computation, http://xxx.lpthe.jussieu.fr/archive/quant-ph/9503016