Chapitre 4 : Complexité quantique

Informatique quantique pour la recherche opérationnelle
Dimitri Watel - ENSIIE

2022

1 Introduction

On s'intéresse dans ce chapitre à plusieurs questions sur la puissance des ordinateurs quantique et leur faisabilité.

2 Complexité d'un algorithme

2.1 Algorihme classique

La complexité (en temps) d'un algorithme classique mesure le nombre d'opérations élémentaires nécessaires pour qu'il se termine. On entend par opération élémentaire les opérations classiques dont on suppose qu'elles se font en un pas de temps constant. Elles sont généralement basées sur les opérations les plus rapides d'un processeur: affectation, lecture dans un tableau, comparaison, opération arithmétique, génération d'un nombre aléatoire, ...

Un algorithme sera dit polynomial (respectivement exponentiel) si ce nombre d'opérations évolue polynomialement (respectivement exponentiellement) avec la taille de l'entrée. On note alors $O(n^c)$ (respectivement $O(2^{n^c})$) sa complexité où n est la taille de l'entrée et où c est une constante.

2.2 Algorihme quantique

L'informatique quantique ajoute à ce panel quelques opérations élémentaires: ajout d'une porte de taille fixe à un circuit; initialisation d'un qbit; passage d'un n-qbit dans une porte, quel que soit n; mesure d'un qbit. La complexité d'un circuit se mesure donc

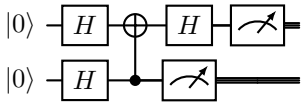
- en nombre de portes m : temps de construction
- en nombre de qbits n : temps d'initialisation
- en profondeur p : temps d'exécution

Un algorithme quantique est un algorithme classique qui construit et exécute un ou plusieurs circuits quantiques. Un algorithme qui exécute t opérations élémentaires classiques et q fois k circuits de taille (m, n, p) a une complexité

$$O(t + q(p + n) + k \cdot m)$$

Cette définition peut varier selon les modèles d'ordinateurs quantiques qu'on utilise.

Par exemple, si on considère le circuit suivant.



Supposons qu'il est exécuté 4 fois et qu'on renvoie la somme des 8 bits mesurés (2 bits par exécution), alors on a $m = 6$, $n = 2$, $p = 4$, $q = 4$ et $k = 1$. Le nombre d'opérations classiques sont les 8 opérations de somme (9 si on compte le fait de renvoyer le résultat comme une opération élémentaire). On aurait donc une complexité de $9 + 4 \cdot 6 + 1 \cdot 6 = 39$.

3 Avantage quantique

Dans cette partie, on va démontrer qu'il existe un problème pour lequel l'informatique quantique apporte une accélération exponentielle.

3.1 Le cas de l'algorithme de Deutsch-Jozsa

Cet algorithme démontre un premier cas d'avantage quantique, mais nous verrons qu'il n'est pas satisfaisant.

On rappelle que cet algorithme permet, ayant accès à une fonction $f : \{0,1\}^n \to \{0,1\}$ dont on sait qu'elle est soit constante soit équilibrée (la moitié des sorties valent 1 et l'autre moitié vaut 0), de déterminer si f est constante.

Lemme 3.1. *Si aucune information n'est donnée sur f, il n'existe pas d'algorithme classique non probabiliste dont la complexité soit meilleure que $\Omega(2^{n-1}t)$, où t est la complexité du calcul de f.*

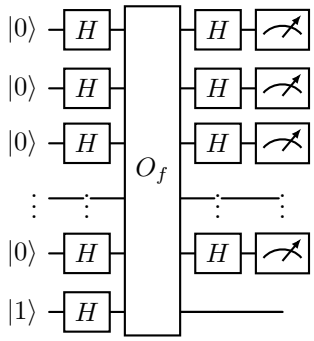
Proof. Nous allons montrer qu'il faut au moins appeler la moitié des entrées pour savoir si la fonction est équilibrée.

Supposons qu'il existe un algorithme $\mathcal{A}$ qui effectue au plus 2^{n-1} appels à f. Soit f_1 une fonction constante. On suppose que $\mathcal{A}$ appelle, dans cet ordre, la fonction f_1 sur $x_1, x_2, \ldots, x_{2^{n-1}}$, puis conclue que f_1 est constante. Puisque f_1 est constante, on a $f_1(x_1) = f_1(x_2) = \cdots = f_1(x_{2^{n-1}})$.

Supposons qu'on donne à $\mathcal{A}$ une fonction équilibrée f_2 qui coincide avec f_1 sur ces 2^{n-1} valeurs.

Puisque $\mathcal{A}$ n'est pas probabiliste et n'a aucune information sur f, alors son exécution ne peut différer quand $f = f_1$ ou $f = f_2$. Elle appelera donc f_2 sur les mêmes entrées que f_1 et prendra les mêmes décisions ; elle concluera donc que f_2 est constante. Ce qui est une contradiction. □

On rappelle que l'algorithme de Deutsch-Josza consiste à créer le circuit suivant, à l'exécuter une fois et à vérifier si tous les qbits en sortie mesurent 0.



On a donc le résultat suivant

Lemme 3.2. *En supposant que la porte O_f a un nombre de porte et une profondeur en $O(t)$, il existe un algorithme quantique résolvant le problème en $O(t+n)$.*

Théorème 3.1. *Il existe un problème que l'informatique quantique peut résoudre exponentiellement plus rapidement que l'informatique classique non probabiliste.*

Ce théorème est toutefois faux pour l'informatique classique au sens large si on considère le résultat suivant:

Lemme 3.3. *Pour tout $\varepsilon > 0$, il existe un algorithme probabiliste qui répond correctement avec une probabilité meilleure que $1-\varepsilon$ dont la complexité est $O(t+n)$.*

Proof. Cet algorithme consiste simplement à tester f sur k valeurs aléatoires, pour $k = \log(\frac{1}{\varepsilon}) + 1$. Si f renvoie k fois la même valeur, on conclue qu'elle est constante. □

Certes, cet algorithme n'est pas exact, mais on peut l'approcher arbitrairement proche d'un algorithme exact. La probabilité qu'il se trompe est quasi-nulle. Il ne semble donc pas y avoir d'avantage quantique pour ce problème.

3.2 Problème de Bernstein Vazirani

On rappelle que ce problème consiste à deviner un entier $s \in \{0,1\}^n$ en ayant accès uniquement à la fonction f qui à $x \in \{0,1\}^n$ renvoie $x \odot s$.

On peut noter qu'il faut $O(n)$ opérations pour calculer f. L'oracle O_f a donc $O(n)$ portes et une profondeur $O(1)$.

Le circuit quantique utilisé par l'algorithme de Bernstein Vazirani est le même que celui de l'algorithme de Deutsch-Jozsa, il a donc également une complexité en $O(t+n) = O(2n) = O(n)$. Cependant, il existe un algorithme en $O(n^2)$ pour résoudre ce problème avec l'informatique classique: il suffit d'appeler f sur chacun des vecteurs de la base pour trouver chaque coordonnée de s. Il n'y a donc clairement pas d'avantage quantique. Mais on peut aller plus loin.

3.2.1 Recursive Fourrier Sampling

Considérons une généralisation du problème de Bernstein et Vazirani.

Soient $2^n + 1$ nombres binaires secrets $s \in \{0,1\}^n$ et $r_x \in \{0,1\}^n$ pour $x \in \{0,1\}^n$. On dispose de deux fonctions $f(x,y) = \underline{r_x} \odot \underline{y}$ et $g(x,y)$ telle que $g(x, r_x) = \underline{s} \odot \underline{x}$ et $g(x, y \neq r_x) = 0$. On souhaite trouver s.

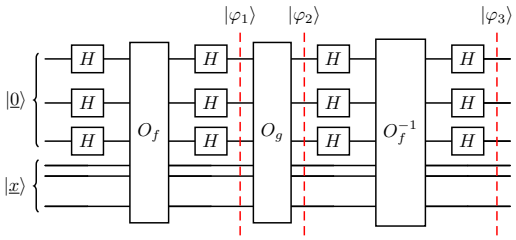
On peut appliquer une technique similaire à la précédente pour construire un algorithme classique qui résout ce problème en $O(n^3)$.

Soit $i \leq n$, si e_i est le i^e vecteur de base, alors on peut retrouver la i^e coordonnée de s en calculant $g(e_i, r_{e_i})$. Mais pour cela, il nous faut r_{e_i}. Pour retrouver r_{e_i}, il suffit de résoudre le problème de Bernstein Vazirani avec la fonction $y \to f(e_i, y) = r_{e_i} \odot y$. On a vu précédemment qu'on pouvait résoudre ce problème en $O(n^2)$. Puisqu'on doit recommencer l'opération pour chaque valeur de i, on a un algorithme en $O(n^3)$.

On va montrer qu'on dispose toujours d'un algorithme quantique en $O(n)$.

Lemme 3.4. *Il existe un circuit quantique qui, avec $|\underline{0}\rangle \otimes |\underline{x}\rangle$, 2 appels à f et 1 appel à g renvoie le qbit $(-1)^{s \odot x} |\underline{0}\rangle \otimes |\underline{x}\rangle$.*

Proof. Le circuit est le suivant

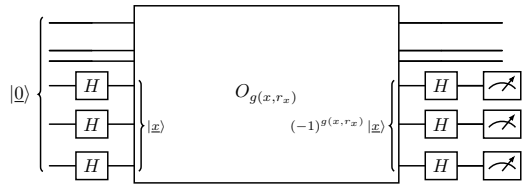


On a

$$|\varphi_1\rangle = \left|\underline{r_x}\right\rangle \otimes |\underline{x}\rangle$$
$$|\varphi_2\rangle = (-1)^{g(x,r_x)} \cdot \left|\underline{r_x}\right\rangle \otimes |\underline{x}\rangle$$
$$|\varphi_3\rangle = (-1)^{g(x,r_x)} \cdot |\underline{0}\rangle \otimes |\underline{x}\rangle$$

□

Ainsi, en appelant $O_{g(x,r_x)}$ le circuit précédent, le circuit suivant résout notre problème en $O(n)$.



On peut voir qu'on a multiplié la profondeur et le nombre de portes par 3.

Nous n'avons toujours pas d'avantage quantique mais on a un peu écarté la complexité classique et la complexité quantique.

3.2.2 Plus de niveaux de récursion

On peut encore plus généraliser le problème récursivement, au lieu d'avoir deux fonctions f et g, on a i fonctions.

$$\begin{aligned} f_i(x_1, x_2) &= s \odot x_1 \text{ si } x_2 = r_{x_1} \\ f_{i-1}(x_1, x_2, x_3) &= r_{x_1} \odot x_2 \text{ si } x_3 = r_{x_1,x_2} \\ f_{i-2}(x_1, x_2, x_3, x_4) &= r_{x_1,x_2} \odot x_3 \text{ si } x_4 = r_{x_1,x_2,x_3} \\ &\dots \\ f_2(x_1, x_2, \dots, x_i) &= r_{x_1,x_2,\dots,x_{i-2}} \odot x_{i-1} \\ &\text{si } x_i = r_{x_1,x_2,\dots,x_{i-1}} \\ f_1(x_1, x_2, \dots, x_i) &= r_{x_1,x_2,\dots,x_{i-1}} \odot x_i \end{aligned}$$

Il faut résoudre un problème de Bernstein Vazirani de niveau $n-1$ pour avoir les informations nécessaires pour trouver s avec la dernière fonction.

On pose $i = \log(n)$.

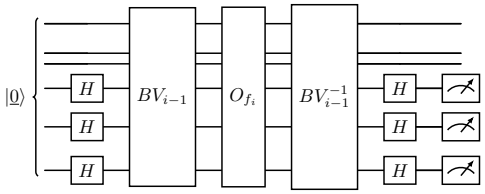
Lemme 3.5. *Il n'existe pas d'algorithme probabiliste pour ce problème dont la complexité soit meilleur que* $\Omega(n^{\log(n)})$.

L'idée de la preuve de ce lemme est de montrer qu'on ne peut obtenir r_x^i avec moins de n appels à f_i.

Lemme 3.6. *Il existe un algorithme quantique pour ce problème dont la complexité est* $O(n2^{\log(n)}) = O(n^2)$.

Proof. On peut le résoudre récursivement. On suppose qu'on dispose d'un algorithme quantique BV_{i-1} qui résout le problème de Recursive Fourrier Sampling avec $i-1$ niveaux.

En généralisant le la technique présentée dans la section précédente, on peut résoudre le problème de niveau i avec le circuit BV_i suivant:



On obtient un algorithme quantique BV_i qui résout le problème avec i niveaux et dont le nombre de portes est deux fois le nombre de portes de BV_{i-1} plus $O(n)$ et dont la profondeur est 2 fois la profondeur de BV_{i-1} plus 3. Une récurrence montre rapidement que la complexité de BV_i est $O(2^i \cdot n)$. Puisque $i = \log(n)$, alors on a bien le résultat souhaité. □

Il existe donc bien un problème pour lequel l'informatique quantique répond exponentiellement plus vite que tout algorithme probabiliste.

4 Classification des problèmes de décision

La classification permets de comparer les problèmes entre eux en fonction de la capacité de l'informatique déterministe, probabiliste ou quantique à résoudre ces problèmes. On va définir ici quelques unes de ces classes et les relations qui les relient.

Toutes les classes définient ici supposent que la réponse est OUI ou NON et que l'information sur l'entrée est totale.

4.1 Classes déterministes non probabilistes

Définition 1. On note P la classe des problèmes que l'on peut résoudre en temps polynomial $(O(n), O(n^2), \dots)$.

On s'est intéressé jusqu'ici au temps mais on peut également s'intéresser à l'espace mémoire nécessaire pour terminer l'algorithme.

Définition 2. On note $PSPACE$ la classe des problèmes que l'on peut résoudre en espace polynomial $(O(n), O(n^2), \dots)$.

Puisqu'il faut du temps pour stocker des données dans la mémoire, on sait que si un algorithme résout un problème en temps polynomial alors il le résout également en espace polynomial.

Théorème 4.1. $P \subseteq PSPACE$

4.2 Classe probabiliste

Définition 3. On note BPP la classe des problèmes pour lesquels il existe un algorithme probabiliste polynomial qui répond correctement avec une probabilité au moins $\frac{2}{3}$.

L'algorithme peut se tromper mais il ne doit pas le faire trop souvent. En répétant cet algorithme de nombreuses fois, on obtient statistiquement le bon résultat plus de fois que le mauvais.

On peut placer BPP entre P et $PSPACE$.

Théorème 4.2. $P \subseteq BPP \subseteq PSPACE$

Proof. Puisque les algorithmes déterministes n'utilisent pas d'aléatoire, alors on peut les considérer comme des algorithmes probabilistes qui ignorent les nombres aléatoires générés et qui répondent correctement avec une probabilité 1, donc supérieure à $\frac{2}{3}$. Un problème dans P est donc également dans BPP.

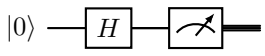
Considérons maintenant un problème dans BPP, il existe donc un algorithme $\mathcal{A}$ probabiliste polynomial qui répond correctement avec une probabilité au moins $O(\frac{2}{3})$. Supposons pour simplifier que $\mathcal{A}$ ne génère qu'un entier aléatoire $r \in \mathbb{N}$. Puisque l'algorithme est polynomial alors r n'est pas *trop grand*, il est borné par une valeur K. Considérons l'algorithme $\mathcal{B}$ qui exécute K fois l'algorithme $\mathcal{A}$, une fois pour chaque valeur i de 1 à K, mais en remplaçant la génération aléatoire de r par $r \leftarrow i$. On compte ensuite le nombre de fois où l'algorithme répond OUI et où il répond NON. La bonne réponse apparait au moins $\frac{2}{3}$ de fois. Donc si $\mathcal{B}$ renvoie la réponse majoritaire, l'algorithme répond correctement à coup sûr. On peut noter que $\mathcal{B}$ n'utilise pas plus d'espace que $\mathcal{A}$, sauf pour compter les réponses OUI et NON. Il fonctionne donc en espace polynomial. Le problème est donc également dans PSPACE. □

4.3 Classe quantique

Les mesures quantiques étant probabilistes, les classes quantiques sont des extensions des classes probabilistes.

Définition 4. On note BQP la classe des problèmes pour lesquels il existe un algorithme quantique polynomial qui répond correctement avec une probabilité au moins $\frac{2}{3}$.

Le théorème suivant se démontre en utilisant le circuit suivant comme générateur aléatoire



Théorème 4.3. $BPP \subseteq BQP$

On prouvera le lien avec PSPACE ultérieurement.

Théorème 4.4. $BQP \subseteq PSPACE$

4.4 Classes non déterministes

Il existe, en complexité classique une classe particulière nommée NP. On peut étendre cette classe aux cas probabiliste et quantique.

On peut définir toutes les classes non déterministe avec une analogie où Merlin et Arthur, les deux figures mythologiques de Camelot, discutent. Arthur veut résoudre le problème. Merlin peut l'aider en lui donnant un indice ; Merlin dispose d'une puissance infinie et connait la réponse. Mais Arthur ne peut le croire sur parole et doit vérifier ses dires. Merlin ne peut donc lui répondre seulement OUI ou NON.

Définition 5. Un problème est dans NP si,

- quand la réponse est OUI, il existe un message de Merlin tel que Arthur comprendra le message et répondra OUI en temps polynomial.
- quand la réponse est NON, pour tout message de Merlin, Arthur comprendra le message et répondra NON en temps polynomial.

Vous trouverez plus d'information sur NP, en particulier pourquoi cette classe est dite non déterministe, en vous intéressant aux machines de Turing. Ce point ne sera pas abordé ici.

On peut étendre facilement cette définition aux deux autres catégories. On note MA la version probabiliste de NP et QMA la version quantique.

Définition 6. Un problème est dans MA si,

- quand la réponse est OUI, il existe un message de Merlin tel que Arthur comprendra le message et répondra OUI en temps polynomial avec une probabilité au moins $\frac{2}{3}$.
- quand la réponse est NON, pour tout message de Merlin, Arthur comprendra le message et répondra NON en temps polynomial une probabilité au moins $\frac{2}{3}$.

Définition 7. Un problème est dans QMA si,

- quand la réponse est OUI, il existe un message (qui peut être sous forme de qbits) de Merlin tel que Arthur comprendra le message et répondra OUI en temps polynomial avec une probabilité au moins $\frac{2}{3}$.
- quand la réponse est NON, pour tout message (sous forme de qbits ou non) de Merlin, Arthur comprendra le message et répondra NON en temps polynomial avec une probabilité au moins $\frac{2}{3}$.

On observe les mêmes relations entre les classes déterministes et non déterministes:

Théorème 4.5. $NP \subseteq MA \subseteq QMA \subseteq PSPACE$

De plus, toute classe déterministe est incluse dans son pendant non déterministe. Il suffit pour Arthur d'ignorer le message de Merlin et on retrouve les définitions déterministes.

Théorème 4.6.
$P \subseteq NP$
$BPP \subseteq MA$
$BQP \subseteq QMA$

4.5 Exemple de problèmes dans ces classes

- Problèmes dans P : Savoir si une liste est triée, Chercher une plus court chemin dans un graphe, Construire un flot maximum, Résoudre un programme linéaire, ...
- Problèmes dans BPP : Soient 3 circuits algébriques, est-ce que exactements deux de ces circuits produisent les mêmes résultats ?

  On pourrait vouloir dire que le problème de Deutsch-Jozsa est dans BPP. Cependant, il s'agit d'un problème à oracle pour lequel l'information en entrée est incomplète. Toutes les classes définies ici supposent qu'on a une information complète sur l'entrée. Il s'agit donc d'un cas un peu particulier. On a supposé dans le lemme 3.1 que l'algorithme ne disposait d'aucune information sur f. Comment pourrait-on disposer de l'oracle sans avoir la moindre information sur f ? Cela semble peu réaliste. En disposant du circuit de O_f, on peut imaginer qu'un algorithme puisse déduire si f est constante ou équilibrée.
- Problème dans BQP : on pourrait vouloir dire que le problème de Bernstein Vazirani appartient à BQP mais, pour les même raisons que précédemment, on dispose d'un problème à oracle avec information incomplète. En ayant le circuit de l'oracle sous les yeux, on devrait avoir des informations sur s.

  Autre cas intéressant : l'algorithme de Shor et la factorisation d'entiers. Il s'agit bien d'un algorithme à information complète et qui semble respecter la définition de BQP. Cependant il ne s'agit pas d'un problème dont les réponses sont OUI ou NON.
- Problème dans QMA : (DIST) : Soit un opérateur quantique Φ, existe-t-il deux entrées $|\underline{x}\rangle$ et $|\underline{y}\rangle$ dont les sorties $\Phi(|\underline{x}\rangle)$ et $\Phi(|\underline{y}\rangle)$ sont *distinguables* ? ; (ID) : Soit un circuit quantique C, est-ce que C produit un résultat *distinguable* de l'identité ?

  Deux qbits sont dits distinguables si la norme de leur différence est élevée. C'est à dire que, statistiquement, en répétant la mesure de ces qbits, on va pouvoir les différencier.

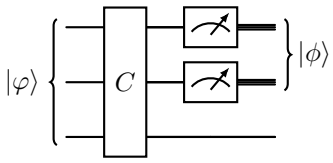
  Deux circuits sont dits distinguables s'il existe une entrée qui, donnée à ces deux circuits, produits deux sorties distinguables.

Il existe une multitude de classes non mentionnées ici. Le lecteur intéressé pour aller chercher des informations sur les classes suivantes:

- RP, PP, RQP, PQP
- ZPP, ZQP, EQP
- BPP-Path
- AM ou AM[k], QAM, QAM[k]
- P-CTC et BQP-CTC (voyage dans le temps)

5 Simulation efficace d'un algorithme quantique

Un point qui n'a pas été développé dans la question précédente est la simulation d'algorithmes quantiques. Un ordinateur quantique peut-il être infiniment puissant ? La réponse est non puisqu'on va montrer qu'il est possible de simuler un algorithme quantique avec un espace polynomial. On considère dans la suite ce circuit où on souhaite mesurer $|\phi\rangle$.



5.1 Complexité de la simulation

Une première question à se poser est: de quoi parle-t-on quand on parle de la complexité de la simulation. Intuitivement, c'est la complexité de l'algorithme classique qui simule le circuit quantique. On va donc compter les opérations élémentaires comme définit en tout début de cours.

La complexité dépend de la taille de l'entrée. Ici l'entrée est le circuit qu'on va simuler. Il faut donc compter sa taille. Formellement, cette taille est l'espace nécessaire pour décrire le qbit $|\varphi\rangle$ et pour décrire le circuit C (par exemple la somme de la taille des portes). La forme du qbit est importante. Il faut plus de place pour décrire $(\frac{1}{p^3}+i2^p)\sum_{x=0}^{2^n-1}|\underline{x}\rangle$ que pour décrire $|\underline{0}\rangle$. De même, il faut plus de place pour décrire la porte de Toffoli que pour décrire la porte X.

Pour simplifier, ici, on supposera que toutes les portes sont de taille 2, que $|\varphi\rangle = |\underline{0}\rangle$ et on ne s'intéressera qu'aux paramètres m, n, p de la taille du circuit C.

5.2 Simulation naive

Pour simuler un circuit, il suffit de calculer le qbit de sortie $|\phi\rangle$. On effectue donc m calculs pour calculer les

qbits successifs du circuit, en commençant par $|\varphi\rangle$. A chaque passage dans une porte, on effectue un calcul en $O(2^n)$. Cette simulation n'est pas très performante puisqu'on manipule un espace exponentiel pour stocker tous les qbits successifs.

Lemme 5.1. *Il est possible de simuler un circuit quantique en temps $O(2^n \cdot m)$ avec un espace exponentiel $O(2^n)$.*

5.3 Simulation efficace

La simulation précédente occulte une chose importante. Notre objectif n'est pas $|\phi\rangle$, notre objectif est de simuler la mesure de $|\phi\rangle$. Avant d'aller plus loin, comment ferions nous pour simuler cette mesure tout en connaissant $|\phi\rangle$?

On utiliser l'algorithme suivant qui simule un tirage aléatoire dans une liste où chaque élément à une certaine probabilité d'être tiré.

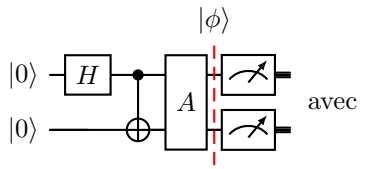
1: $r \leftarrow$ un nombre aléatoire entre 0 et 1
2: **Pour** x de 0 à $2^n - 1$ **Faire**
3: $\quad p_x \leftarrow |\langle \underline{x}|\phi\rangle|^2$ // La probabilité de mesurer x.
4: $\quad r \leftarrow r - p_x$
5: $\quad$ **Si** $r < 0$ **Alors**
6: $\quad\quad$ **Renvoyer** x

La somme des probabilité vaut $1 > r$, donc il est certain que cet algorithme renvoie un résultat. La probabilité qu'il renvoie x est exactement p_x (si la génération de r est parfaite).

On peut se poser une petite question sur la précision nécessaire de r. Elle dépend de la précision des coefficients de $|\underline{x}\rangle$ dans $|\phi\rangle$. Plus le vecteur $|\phi\rangle$ est précis, plus r doit être précis. Sans rentrer dans les détails, la précision de $|\phi\rangle$ augmente faiblement avec la précision de $|\varphi\rangle$ et le nombre de portes dans le circuit. Cette précision n'est donc pas un gros problème.

Le point d'intérêt ici est de remarquer qu'on a pas besoin de tous les p_x en même temps. On peut calculer p_0. Faire une boucle de cet algorithme, puis calculer p_1, faire une boucle, et ainsi de suite. A chaque boucle, on peut jeter la dernière variable p_x car elle ne servira plus. Bref, si on est capable de calculer, pour un $x \in [\![0; 2^n - 1]\!]$ donné en particulier, la probabilité p_x en espace polynomial, alors on aura une simulation de la mesure de $|\phi\rangle$ en espace polynomial (mais on n'aura jamais $|\phi\rangle$ lui-même en mémoire).

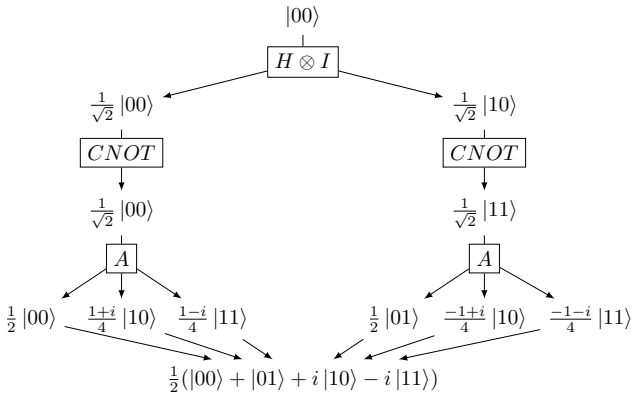
Considérons un exemple



avec

$$A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \frac{1+i}{2} & \frac{-1-i}{2} & \frac{1-i}{2} & \frac{-1+i}{2} \\ \frac{1-i}{2} & \frac{-1+i}{2} & \frac{1+i}{2} & \frac{-1-i}{2} \end{pmatrix}.$$

En simulant naïvement le circuit, on obtiendrait l'arbre suivant



$$|\phi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + i\,|10\rangle - i\,|11\rangle)$$

On voit que chaque porte produit au plus 4 sorties, chacune avec un coefficient différent et un état de base différent. C'est parce que chaque porte est de taille 2. A chaque étape, on prend un état de base, on le passe dans la porte suivante ; ce qui crée 4 états superposés au maximum en sortie ; et on recommence. A la fin, on additionne tous les coefficients associés aux mêmes états de base et on retrouve $|\phi\rangle$. Ce calcul est tout aussi peu performant que la simulation naïve, l'arbre est de taille $O(4^m)$. Mais on pourrait explorer cet arbre sans le garder en mémoire entièrement. On peut faire un parcours en profondeur à gauche. A chaque fois qu'on arrive sur une feuille, on rajoute un coefficient devant le bon état dans le vecteur $|\phi\rangle$, puis on remonte l'arbre pour aller explorer les feuilles plus à droite. Cette exploration n'est pas plus rapide mais utilise un peu moins d'espace. Cependant, on maintient toujours un vecteur $|\phi\rangle$ en mémoire.

On rappelle qu'on a fixé un $x \in [\![0; 2^n - 1]\!]$ et qu'on cherche p_x. Si on a $x = 3$ par exemple, on chercherait la probabilité de mesurer $|11\rangle$. On pourrait lors du parcours de l'arbre ignorer tout ce qui ne concerne pas $|11\rangle$. On verrait en premier $\frac{1}{2}|00\rangle$. On ne le garde pas en mémoire, car on cherche à mesurer $|11\rangle$. On voit ensuite $\frac{1+i}{4}|10\rangle$. On ne le garde pas non plus en mémoire. Puis, on a $\frac{1-i}{4}|11\rangle$. On le garde en mémoire et on continue. On voit ensuite $\frac{1}{2}|01\rangle$ et $\frac{-1+i}{4}|10\rangle$ qu'on ne conserve pas ; puis

$\frac{-1-i}{4}|11\rangle$ qu'on mélange au coefficient $\frac{1-i}{4}$ qu'on a gardé en mémoire. On obtient alors $\frac{-i}{2}$. On arrive ensuite à la fin de l'arbre, donc le coefficient obtenu donne une probabilité de $p_3 = |\frac{-i}{2}|^2 = \frac{1}{4}$. A tout instant, on n'a gardé en mémoire deux choses: la somme des coefficients devant $|11\rangle$ qu'on a vus ; et la branche de l'arbre où on se trouve pour pouvoir continuer à le parcourir en profondeur. Cette branche a une taille égale aux nombres de portes m. On a donc bien gardé un espace polynomial, environ $O(m)$, tout au long de l'algorithme.

5.4 Cas des grandes portes

On a fait l'hypothèse que les portes sont de taille 2. Cette hypothèse nous simplifie la tâche car, dans l'arbre, chaque porte va créer au plus 4 nouvelles branches (pour les au plus 4 états superposés en sortie la porte).

Pour une porte de taille k quelconque, on aura 2^k nouvelles branches. Cela ne change pas l'algorithme, il devient un peu plus long car l'arbre est plus long à parcourir. Si la porte la plus grande est de taille n, alors l'arbre est de taille $O((2^n)^m) = O(2^{nm})$ ce qui reste exponentiel en la taille du circuit. La complexité en espace reste bien $O(m)$ pour stocker la branche courante de l'arbre.

On peut se dire que l'espace n'est pas polynomial puisqu'il faut déjà $O(2^n)$ cases pour stocker le circuit. C'est vrai, mais on rappelle que la complexité se calcule vis-à-vis de la taille du circuit. Si le circuit prend une place N alors tout algorithme qui utilise un espace $O(N)$ est linéaire vis-à-vis de la taille du circuit, quelle que soit la valeur de N. Donc l'espace n'est pas polynomial vis-à-vis de n mais n n'est pas la taille du circuit.

5.5 Cas d'un vecteur $|\varphi\rangle$ superposé

Si l'entrée est un état superposé, il suffit de dupliquer l'arbre pour chaque état de base de $|\varphi\rangle$. On garde donc bien un espace polynomial. Si décrire $|\varphi\rangle$ nécessite déjà un espace exponentiel, alors on est dans le même cas de figure que pour les grandes portes.

6 Portes universelles

Le cas des grandes portes peut être problématique d'un point de vue pratique. Tout algorithme quantique devrait (on l'espère) fonctionner sur un vrai ordinateur quantique. Pour cela l'ordinateur doit pouvoir simuler n'importe quel circuit. Mais clairement, un ordinateur ne peut implanter toutes les portes quantiques puisqu'il y en a une infinité. On peut cependant avoir l'espoir de simuler toute porte quantique avec un circuit constitué de petites portes ; comme on a pu le faire par exemple avec la porte S de l'algorithme de Grover ou la porte F de la transformée de Fourrier dans l'algorithme de Shor. On dit que l'ensemble de petites portes choisi est **universel** s'il peut simuler n'importe quelle autre porte.

C'est le cas en classique où la porte NAND est capable de simuler n'importe quelle autre porte logique. Un processeur n'a donc, théoriquement, besoin que de portes NAND.

On va commencer par un résultat négatif qui sera vite succédé par un résultat positif.

Théorème 6.1. *Pour chaque ensemble fini P de porte, il existe une porte quantique A telle que A ne peut être décomposée en un circuit n'utilisant que les portes de P.*

Proof. La preuve est assez simple : l'ensemble des portes quantiques, donc des matrices unitaires à coefficients complexes, est non dénombrable. En effet, la porte de changement de phase suivante est unitaire:

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \text{ où } \phi \in [0, 2\pi]$$

Puisque l'intervale $[0, 2\pi]$ n'est pas dénombrable, alors ces portes ne sont pas dénombrables.

Cependant, si P est fini, l'ensemble des circuits qu'on peut construire avec P est dénombrable. □

Donc pour être universel, notre ensemble de porte doit être infini, voire indénombrable. On va donc restreindre notre définition d'universalité.

Définition 8. Soit $\varepsilon > 0$, on dit que le circuit C approche la porte A à ε près si, pour tout $|\varphi\rangle$, on a $\|C|\varphi\rangle - A|\varphi\rangle\| \leq \varepsilon$.

Définition 9. Soit $\varepsilon > 0$, on dit qu'un ensemble P de portes est universel si et seulement si, pour toute porte quantique A, on peut approcher A, à ε près, avec un circuit quantique ne contenant que des portes de P.

Théorème 6.2. *Les ensembles suivants sont universels:*

- *H et Toffoli (portes non complexes)*
- *$CNOT$ et toutes les portes sur 1 qbit : $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$, $P(\varphi)$ pour tout θ et tout φ*
- *$CNOT$, H, $S = \sqrt{Z}$ et $T = \sqrt[4]{Z}$*
- *$CNOT$, $R_y(\pi/4)$, $P(\pi/2)$*
- *Avec une porte A sur 2 qbit choisie uniformément, $CNOT$, A, $P(\pi/2)$ a une forte probabilité d'être un ensemble universel.*

On peut noter que le nombre d'ensembles de portes universelles est lui-même non dénombrable.

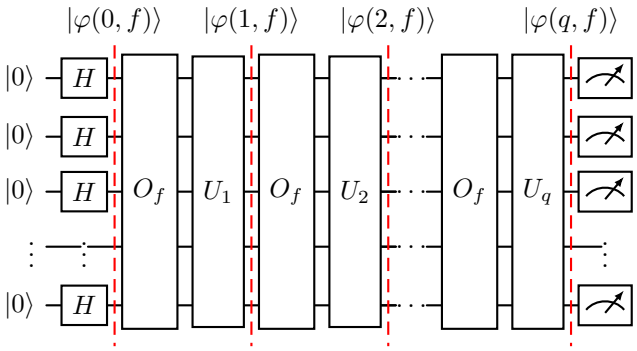
On peut se poser plusieurs petites questions:

- Combien de portes faut-il pour simuler A ? Si A est de taille 2^n il faut environ $O(4^n)$ portes pour simuler A.
- Comment H et Toffoli pourraient être universelles alors qu'elles ne sont pas complexes ? Ces deux portes simulent toutes les portes quantiques à coefficients réels. Un point important ici est que tout calcul quantique avec des coefficients complexes peut être simulé par des portes quantiques à coefficients réels et utilisant un qbit de calcul supplémentaire.
- Comment prouver l'universalité ? C'est de loin le point le plus compliqué. Une technique consiste à utiliser le théorème de Solovay–Kitaev : si un ensemble P de portes de taille 1 génère un ensemble dense dans l'ensemble de toutes les portes de taille 1 ; alors P est CNOT forment un ensemble universel.

7 Optimalité de Grover

Une dernière question importante en algorithmique est "Aurais-je pu faire mieux ?". On a démontré que l'algorithme de Grover était capable, connaissant une fonction f, de trouver un élément s tel que $f(s) = 1$, en $O(\sqrt{2^n})$ appels à f. On peut se demander s'il existe un algorithme qui ferait la même chose avec moins d'appels à f.

Pour cela, on va supposer l'existence du circuit suivant:



Grover est un cas particulier de ce circuit où la porte S, l'opérateur de Grover, remplace toutes les portes U_i. On pourrait se dire qu'en utilisant d'autres portes que la porte S, on pourrait converger vers le résultat plus vide. On peut démontrer que non, pour cela, on va donner une borne inférieure à q.

On note f_s la fonction telle que $f(s) = 1$ et $f(x \neq s) = 0$. Si $y \neq z$ alors on sait que $|\underline{y}\rangle \perp |\underline{z}\rangle$. Or, on sait que $|\varphi(q, f_y)\rangle \simeq |\underline{y}\rangle$ et $|\varphi(q, f_z)\rangle \simeq |\underline{z}\rangle$. De plus, on sait que $|\varphi(0, f_y)\rangle = |\varphi(0, f_z)\rangle$ puisqu'il s'agit dans les deux cas de $H^{\otimes n} |\underline{0}\rangle$.

On voit donc que le circuit part du même qbit mais converge dans deux directions opposées. La valeur de q dépend donc de la capacité des portes U_i à éloigner $|\varphi(i, f_y)\rangle$ et $|\varphi(i, f_z)\rangle$ pour tout y et z. Et on peut montrer que cette capacité d'éloignement est assez faible.

Poursuivons le raisonnement un peu plus loin. Que se passerait-il si on donnait au circuit la fonction zr égale à 0 partout ? $|\varphi(i, zr)\rangle$ ne devrait s'approcher d'aucun qbit en particulier. Ce qbit devrait rester parfaitement superposé tout au long du circuit. Dans le cas contraire, cela signifierait que le circuit converge préférentiellement vers un certain y ; et ce serait contreproductif pour les fonction f_z où $z \neq y$.

On va donc démontrer la capacité d'éloignement de f_y et de zr. Cette fonction a l'avantage que l'oracle O_f est égal à l'identité. L'utiliser permet donc de simplifier les calculs. Pour démontrer le résultat, il faut démontrer le lemme suivant (fait en TD).

Lemme 7.1.

$$\sum_{y=0}^{2^n-1} \| |\varphi(q, zr)\rangle - |\varphi(q, f_y)\rangle \| \geq \sqrt{2} \cdot (2^n - 1)$$

$$\sum_{y=0}^{2^n-1} \| |\varphi(q, zr)\rangle - |\varphi(q, f_y)\rangle \| \leq 2\sqrt{2^n} q$$

On obtient immédiatement le théorème suivant:

Théorème 7.1.

$$q = \Omega(\sqrt{2^n})$$