



HAL
open science

Évaluation du potentiel des machines quantiques pour l'optimisation combinatoire

Julien Rodriguez

► **To cite this version:**

Julien Rodriguez. Évaluation du potentiel des machines quantiques pour l'optimisation combinatoire. Informatique [cs]. 2020. dumas-03384577

HAL Id: dumas-03384577

<https://dumas.ccsd.cnrs.fr/dumas-03384577>

Submitted on 19 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0
International License

Évaluation du potentiel des machines quantiques pour l'optimisation combinatoire



Mémoire de fin d'étude

Master *Sciences et Technologies*,
Mention *Informatique*,
Parcours MASTER INFORMATIQUE THÉORIQUE

Auteur

Julien Rodriguez

Superviseur

Eric Bourreau

Lieu de stage

LIRMM UM5506 - CNRS, Université de Montpellier

Résumé

L'arrivée des machines quantiques risque de révolutionner de nombreux domaines, en particulier celui des algorithmes combinatoires. Nous proposons dans ce stage d'évaluer la performance des algorithmes existants (et publiés) et d'éclairer les opportunités des algorithmes quantiques.

Dans un premier temps, nous allons aborder l'algorithme de recherche dans une base non structurée de Lov Grover (1996) ainsi que deux autres algorithmes inspirés de l'algorithme de Grover traitant le problème de satisfiabilité d'une formule logique et la recherche d'éléments dans un tableau.

La seconde partie présente des algorithmes quantiques nécessaires pour faire de l'optimisation combinatoire, tels que la somme ou encore la minimisation. Enfin, nous abordons le problème de la couverture d'arêtes (Vertex Cover - VC) et voyageur de commerce (Traveling Salesman Problem - TSP).

Abstract

Quantum machine provide an opportunity to revolutionizing many fields, especially in combinatorial algorithms. In this internship, we propose to assess the performance of existing (and published) algorithms and clarify the opportunities of quantum algorithms.

In a first step, we use a search algorithm on an unstructured database published by Lov Grover (1996) and two other algorithms inspired by the Grover algorithm dealing with the satisfiability problem on a logical formulas and search for elements in an array.

The second part present quantum algorithms necessary to do combinatorial optimization, such as the sumation and minimization. Finally, we experimentally solve the problem of vertex cover and traveling salesman problem.

Table des matières

Table des matières	v
1 Introduction	1
2 Calcul quantique	3
2.1 Qubits	3
2.2 Portes quantiques	4
3 Algorithmique de recherche	13
3.1 Grover	13
3.2 3 SAT	18
3.3 Exécutions	23
3.4 Recherche dans un tableau	24
4 Optimisation Combinatoire	31
4.1 Recherche du minimum	31
4.2 Minimiser une somme	36
4.3 Couverture de sommets de cardinalité minimale	38
4.4 Voyageur de commerce	42
5 Conclusion	47
Bibliographie	49

Introduction

L'apparition de l'informatique quantique ces dernières années se base sur un nouveau paradigme de calcul permettant d'envisager la résolution de problèmes jusqu'ici inaccessibles. Plusieurs facteurs expliquent cet intérêt croissant : de nouveaux algorithmes en rupture avec l'existant, la limitation de la loi de Moore sur les machines actuelles et la création de machines physiques basées sur la notion de qubits.

En effet, la découverte de l'algorithme de Shor en 1994 a permis pour la première fois de factoriser des grands nombres en temps sous exponentiel. De même la création de qubits assez fiables et l'apparition ou l'amélioration régulière de plusieurs architectures d'ordinateurs quantiques a permis l'arrivée de nouveaux acteurs sur le marché (D-wave, IBM, et récemment Google avec son annonce de suprématie quantique).

Concernant plus précisément l'optimisation combinatoire, les premiers résultats datent de plus de 20 ans et furent principalement publiés dans des journaux de physique. Il a fallu attendre seulement ces 5 dernières années pour voir arriver la construction de machines (et de simulateurs logiciels) permettant de réellement exécuter ces circuits théoriques et obtenir les premiers résultats expérimentaux.

L'objectif de ce stage est donc double car il faut dans un premier temps transformer les algorithmes de physiciens présentés dans les articles en algorithmes d'informaticiens (changement de paradigme) ce qui permet dans un second temps de commencer à évaluer ce qu'apporte les machines quantiques aux problèmes d'optimisations connus en informatique classique, comme SAT ou le voyageur de commerce.

Dans ce mémoire, le premier chapitre présente les bases du calcul quantique. Le second chapitre s'intéresse aux algorithmes de recherche dans une base non-structurée duquel découle les problèmes d'optimisation étudiés. Le chapitre suivant traite certains problèmes d'optimisations étudiés comme la couverture de sommets (Vertex Cover - VC) et le voyageur de commerce (Traveling Salesman Problem - TSP).

Bien que la plupart des "algorithmes" présentés aient déjà été publiés par morceaux (que ce soit officiellement dans des revues, postés sous arxiv, ou simplement accessibles via des github publics), nous proposons pour la première fois leurs "implémentations" homogènes en insistant sur tous les détails laissés en suspens dans les résultats théoriques comme l'accès aux données représentant les instances.

Calcul quantique

Une majorité de problèmes obtiennent une accélération quadratique avec le paradigme du calcul quantique. Ce nouveau paradigme repose essentiellement sur les premiers postulats de la mécanique quantique. Nous allons ici présenter brièvement les briques de base du calcul quantique dans le but d'une meilleure compréhension des propriétés qu'offrent ces machines. Bien que le contexte du calcul quantique ait déjà été expliqué lors du premier rapport, il nous a semblé important de rappeler celui-ci. Le lecteur habitué au circuit logique et autre intrication peut directement sauter 10 pages et aller en section 3.

2.1 Qubits

Qu'est-ce qu'un qubit ?

Pour les ordinateurs classiques, le bit peut prendre deux valeurs, 0 ou 1. L'ordinateur réalise ses calculs grâce à un système physique à deux états (aimantation, interrupteur, condensateurs...).

Le bit quantique ou *qubit* est un système *quantique* pouvant prendre une infinité d'états, en particulier 0 ou 1 comme l'ordinateur classique. Ces états sont notés $|0\rangle$ et $|1\rangle$ par convention introduite par P. A. M Dirac [7], et forment les états de base d'un qubit. Un *qubit* est donc constitué d'une superposition quantique linéaire de ces deux états de base. On dit que tous les états possibles sont simultanément présent, c'est la *superposition*. Tout état $|\Psi\rangle$ (qubit) est représenté comme ceci :

$$|\Psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

avec α_i un nombre complexe.

Le principe de superposition peut être imagé par le célèbre exemple du *chat de Schrodinger*. Dans cet exemple, en l'absence de mesure, le chat est en superposition d'états (mort/vivant).

La mesure

Un qubit peut avoir une multitude de valeurs mais une fois mesurée sa valeur est fixe. Si l'on cherche à *mesurer* un qubit, on trouvera 0 s'il est dans l'état $|0\rangle$ et 1 s'il est dans l'état $|1\rangle$. Si le qubit est dans l'état $|\Psi\rangle$, la mesure va projeter cet état soit sur $|0\rangle$ avec la probabilité $||\alpha_0||^2$, soit sur $|1\rangle$ avec la probabilité $||\alpha_1||^2$. On en déduit que :

$$||\alpha_0||^2 + ||\alpha_1||^2 = 1$$

La projection d'un qubit sur un état $|0\rangle$ ou $|1\rangle$ est définitive après sa lecture. Si le qubit est dans un état $|\Psi\rangle$ et que lors de la mesure on obtient l'état $|1\rangle$, alors toutes autres mesures donneront l'état $|1\rangle$. Les règles décrivant le bit quantique (système quantique) reposent sur les postulats de la mécanique quantique [8]. À partir de ces règles ont été défini des opérations permettant de modifier l'état du qubit en modifiant les valeurs α_0 et α_1 en appliquant des opérations appelées aussi *portes quantiques*. Ces calculs peuvent associer un ou plusieurs qubits, tout cela sans mesurer le ou les qubits ce qui entraînerait une projection irréversible dans l'état $|0\rangle$ ou $|1\rangle$. Après l'application de toutes les portes nécessaires au calcul, on mesure la valeur finale du qubit.

Registre quantique

Un registre quantique est une généralisation d'un système quantique avec un qubit à un système à plusieurs qubits. Par exemple, la notation d'un registre $|\Psi\rangle$ à n qubits sera l'état :

$$|\Psi\rangle = \alpha_0 |0\dots 0\rangle + \alpha_1 |0\dots 01\rangle + \dots + \alpha_N |1\dots 1\rangle$$

Avec $N = 2^n - 1$ et la chaîne binaire décrivant les états tel que 0...01 par exemple, sont de longueur n . Comme le qubit, le registre de qubits détient aussi la propriété de superpositions d'états permettant une projection équiprobable entre tous les états du registre. Ceci est une différence notable avec un registre classique de bits, qui lui ne peut avoir qu'une seule valeur fixe à la fois, comprise entre 0 et $2^n - 1$. Il est donc possible dans un registre quantique de superposer les qubits et de pouvoir traiter les valeurs de l'intervalle discret $[0, 2^n - 1]$ en même temps et de façon déterministe car l'équation de Schrödinger décrivant l'évolution d'un système quantique est déterministe¹ [18]. La mesure du résultat projettera l'état du système dans un unique état, dépendant d'une probabilité d'apparition liée à ses paramètres α .

2.2 Portes quantiques

Les portes quantiques sont les opérateurs nécessaires au calcul quantique. Ils vont modifier l'état de nos qubits en changeant la valeur des coefficients α . Cela a pour effet de modifier la probabilité de la projection de l'état $|\Psi\rangle$ sur les états $|0\rangle$ ou $|1\rangle$.

¹Cette équation est du premier ordre par rapport au temps t et complètement déterministe : les mêmes conditions initiales produisent le même ensemble de solutions, c'est-à-dire la même évolution dans le temps et l'espace[18].

Portes de bases

Les portes quantiques correspondent à des opérateurs unitaires² et sont réversibles. Or, la seule opération non triviale réversible réalisable sur un bit classique est le *NON* logique. En effet : $x_1 \equiv \neg\neg x_1$.

Ce n'est pas un problème car tout algorithme classique peut s'exprimer en un algorithme réversible en rajoutant des bits de calcul supplémentaire [1].







Représentation graphique	Portes
	X (NOT)
	Porte de Hadamard
	La porte Z
	Controlled-NOT (CNOT)
	Porte de Toffoli (CCNOT)
	Mesure l'état d'un qubit

FIGURE 2.1 : Portes quantiques et leurs représentations graphiques

- La porte **X**, ou **NOT** est équivalente au **NON** logique
- La porte de Hadamard provoque la superposition des états $|0\rangle$ et $|1\rangle$. Lors de la mesure on a autant de chance de mesurer $|0\rangle$ et $|1\rangle$, ($\alpha = \beta$)
- La porte **Z** provoque une inversion de phase du qubit : $|0\rangle$ ne change pas mais $|1\rangle$ devient $-|1\rangle$
- La porte *Controlled-NOT* ou **CNOT**, change la valeur du second qubit (applique un **NOT**) si et seulement si le qubit de contrôle est dans l'état $|1\rangle$.
- La porte **CCNOT** ou porte de Toffoli, due à son créateur *Tomaso Toffoli*, est une porte **ternaires** et prend en entrée deux qubits q_1 et q_2 , appelés qubits de contrôle . La porte **CCNOT** se comporte comme la porte **CNOT** mais avec 2 qubits de contrôle. Ici, les qubits de contrôle doivent être dans l'état $|1\rangle$ pour appliquer un **NOT** sur le troisième qubit, appelé : "*qubit cible*"
- La porte **M** mesure (projette) le qubit dans un état.

²En mécanique quantique, l'unitarité désigne le fait que l'évolution de la fonction d'onde au cours du temps doit être compatible avec l'interprétation probabiliste qui lui est associée.

Avec la porte de Toffoli, il est possible de reproduire le **ET** logique. Soit 3 qubits q_1 , q_2 et q_3 , tels que q_1 et q_2 sont superposés et q_3 initialisé dans l'état $|0\rangle$. Si nous appliquons la porte **CCNOT** avec q_1 et q_2 comme qubit de contrôle, alors q_3 sera dans l'état $|1\rangle$ si et seulement si q_1 et q_2 sont tout deux dans l'état $|1\rangle$. Ainsi, il est l'équivalent du **ET**.

Comme nous avons la possibilité de reproduire le **NON** avec la porte **X** et le **ET** avec la porte **CCNOT**, et que le **NON-ET** est universel, il est possible reproduire toutes les fonctions de la logique sur un ordinateur quantique.

Les machines de turing quantiques sont Turing Complet [2].

Portes multi-qubits

Les machines ayant de plus en plus de qubits, il y a un intérêt nécessaire de généraliser les portes à un système quantique sur plusieurs qubits, appelé registre quantique comme en 2.1. Si un tel registre contient n qubits, nous pouvons alors être sur un espace à 2^n états. Par exemple, en appliquant la porte de Hadamard sur tout le registre, nous obtenons une superposition des 2^n états. Il s'agit de cette propriété de manipulation de la superposition d'un nombre d'états exponentiel que nous allons exploiter dans nos programmes.

Un autre aspect est de permettre le calcul de portes contrôlées avec plusieurs qubits. Nous avons vu qu'il était possible de le faire avec la porte **CCNOT** car deux qubits de contrôles sont utilisés. Traitons le cas d'un **N_CNOT** et plus généralement de la porte **N_CU** qui applique la porte **U** si les N qubits de contrôles sont dans l'état $|1\rangle$, avec **U** une porte quelconque, en s'inspirant de Michael Nielsen et Isaac Chuang dans [15]. Cette généralisation ouvre la porte à une modélisation d'instances plus grandes que nous verrons plus loin dans ce document.

Prenons le cas de la **CCNOT** en guise d'exemple pour mieux comprendre l'idée générale. Pour généraliser la porte **CNOT** à trois qubits de contrôle, nous utilisons la porte **CCNOT** pour stocker le résultat d'un **CNOT** entre les deux premiers qubits de contrôle dans un qubit appelé *ancilla*. Puis nous utilisons ce résultat pour calculer un **CNOT** avec le troisième qubit de contrôle. Si l'un des deux premiers qubits de contrôle vaut $|0\rangle$ alors ce $|0\rangle$ sera transmis au second calcul par le qubit *ancilla* qui le transmettra au calcul final. Pour rappel, la **CCNOT** applique un **NOT** sur le qubit cible si tous les qubits de contrôle ont pour valeur $|1\rangle$. Voici un circuit de la **CCNOT** avec 3 qubits de contrôle. Les noms des qubits se trouvent sur la gauche. Le circuit se lit de gauche à droite. Chaque étape représente une opération dans le temps et modifie l'état des qubits.

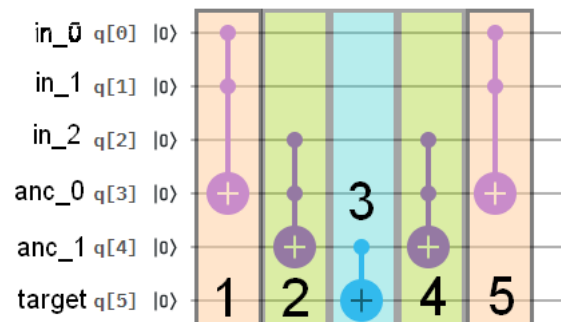


FIGURE 2.2 : CCCNOT

1. Contrôle des qubits $|in_0\rangle$ et $|in_1\rangle$ et on stocke le résultat dans le qubit $|anc_0\rangle$
2. Contrôle du $|in_2\rangle$ avec le résultat précédant et stockage le résultat dans $|anc_1\rangle$.
3. Application d'une **CNOT** entre $|anc_1\rangle$ et le qubit cible $|target\rangle$
4. Réinitialiser le qubit $|anc_1\rangle$ pour une prochaine utilisation
5. Réinitialiser le qubit $|anc_0\rangle$ pour une prochaine utilisation

La série d'opérations effectuée après l'opération **CNOT**($|anc_1\rangle$, $|target\rangle$) est appelée : *unroll*.

En généralisant cette modélisation, nous obtenons un algorithme qui va appliquer le rôle de la porte **CNOT** pour un nombre arbitraire de qubits de contrôle :

Algorithme 1 : n_cnot

```

entrées   : circuit : Circuit quantique, controles : registres de qubits, ancillas :
              registres de qubits, target : qubit
sortie    : circuit : Circuit quantique
1 n = size(controles)
2 si n == 1 alors
3   | circuit.cx(controles[0], target)
4 si n == 2 alors
5   | circuit.cxx(controles[0], controles[1], target)
6 si n > 2 alors
7   | circuit.cxx(controles[0], controles[1], ancillas[0])
8   | pour chaque i de 2 à n faire
9     | circuit.ccx(controles[i], ancillas[i-2], ancillas[i-1])
10  | fin
11  | circuit.cx(ancillas[n-2], target) pour chaque i de n - 1 à 1 faire
12    | circuit.ccx(controles[i], ancillas[i-2], ancillas[i-1])
13  | fin
14  | circuit.ccx(controles[0], controles[1], ancillas[0])
15 retourner circuit

```

Il existe également d'autres portes contrôlées comme la porte **CZ** qui applique la porte **Z** sur le qubit $|target\rangle$ si le qubit de contrôle est dans l'état $|1\rangle$ ou encore la porte **CH** qui admet le même comportement en appliquant la porte **H** sur qubit cible.

À partir de ces portes et de la généralisation de la porte **N_CNOT** vue précédemment, il est possible de réaliser le même schéma avec toutes les portes contrôlées.

Les portes quantiques et les registres permettent de réaliser des circuits quantiques exécutables sur un simulateur ou sur une machine quantique. Le LIRMM bénéficie d'un accès aux machines quantiques d'IBM ainsi que d'un simulateur permettant d'exécuter des circuits. Pour faciliter le développement de circuit quantique et l'utilisation des machines quantiques, IBM lança en 2016 un *framework* python nommé Qiskit [19].

Nous utilisons Qiskit pour écrire nos circuits et les exécuter sur simulateur ou sur des machines physiques disponible via le *cloud*. Chaque machine a sa propre architecture et chaque qubit à son propre taux d'erreur par porte :

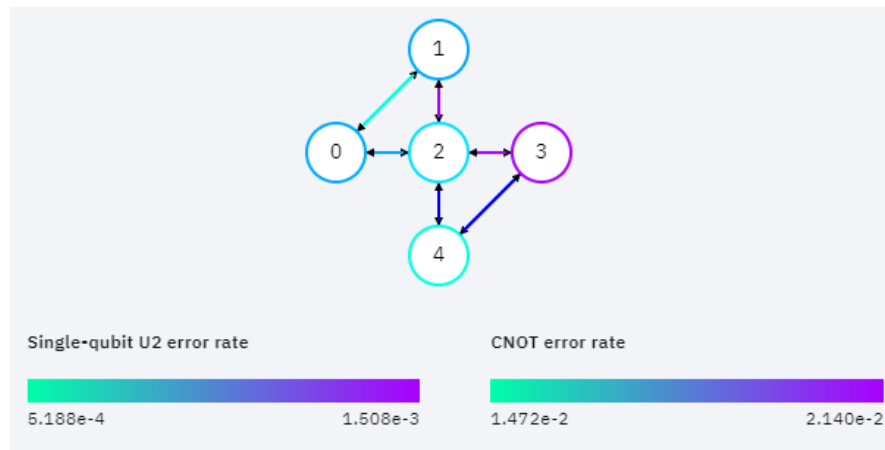


FIGURE 2.3 : Architecture et taux d'erreurs : *ibmq_yorktown*

Notons que sur les machines réelles, seules les opérations **U2**³ et **CNOT** sont présentes. Par une étape appelée *transpilation*, le circuit sera transformé en une succession de **U2** et de **CNOT**.

Quelques exemples

L'exemple, ci-dessous, est un circuit qui va provoquer une superposition de 4 états (2 qubits).

³Cette porte n'est pas présentée car nous ne l'utiliserons pas pour nos circuits. Cette porte change l'angle du qubit sur lequel elle est appliquée

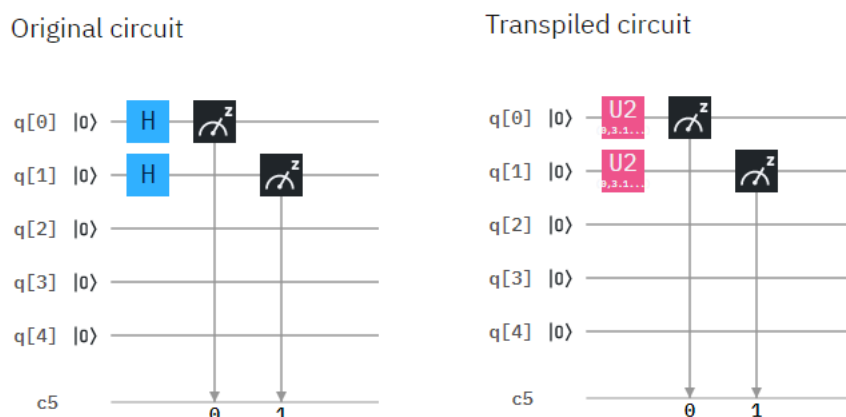


FIGURE 2.4 : Circuit avant et après transpilation

Remarquons également que l'architecture joue un rôle sur le nombre de portes. Par exemple, si nous avons dans le circuit de base une opération entre deux qubits non connectés directement l'un à l'autre, comme illustré sur la figure 2.3, on sera obligé d'échanger des qubits par une opération de *swap*⁴ pour réaliser le calcul physiquement. Bien évidemment, si les qubits du voisinage d'un des deux qubits ne sont pas utilisés, on peut "échanger" les qubits dès l'écriture du circuit.

Après la transpilation, à lieu l'exécution du circuit. Le circuit est exécuté plusieurs fois pour obtenir une meilleure visualisation des résultats. Par exemple, si nous exécutons une seule fois le circuit de la figure 2.4, on obtiendrait qu'une seule projection et donc qu'un seul état du registre comme résultat. Alors que si nous exécutons le circuit plusieurs fois on obtiendra une distribution équiprobable de toutes les solutions possibles (00, 01, 10, 11). Chaque exécution est appelée un *shot*.

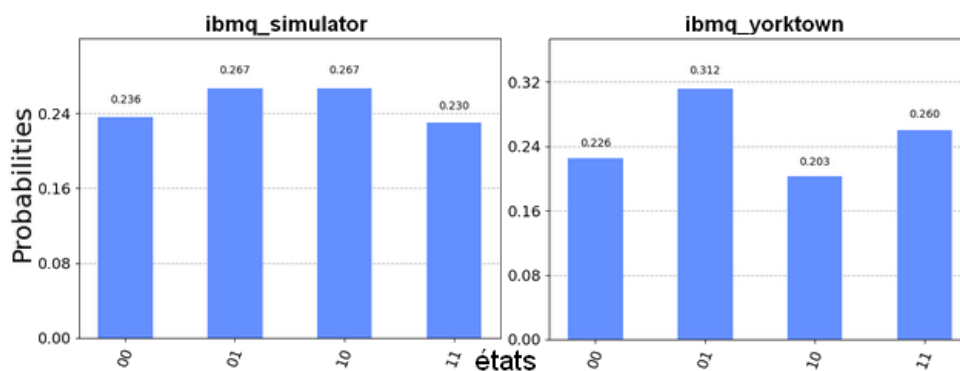


FIGURE 2.5 : Résultats pour l'exécution du circuit de la figure 2.4

Les histogrammes de la figure 2.5 représentent les proportions des résultats obtenus. Pour chaque circuit, le nombre de *shots* était de 1024, donc 1024 exécutions. Nous

⁴L'opération de *swap* permet d'échanger deux qubits par le biais de 3 **CX**.

pouvons voir sur la figure 2.5 que les 4 états possibles sont présents, ils sont **superposés** après le passage des portes **H**, il s'agit du premier phénomène fondamental de la théorie quantique. Sur l'histogramme de droite de la figure 2.5 nous observons le même résultat, mais la répartition est moins homogène. Cela est dû aux taux d'erreur apporté par le bruit (compris entre $1,508.10^{-3}$ et $5,188.10^{-4}$) sur les machines actuelles, illustré sur la figure 2.3. Cela peut s'expliquer par des variations du champs magnétique terrestre, de température au sein du circuit des qubits (pour rappel autour de 0,015K) ou d'interférence entre les fréquences de lecture de chaque qubit.

Le circuit ci-dessous est un exemple d'intrication. Le qubit de contrôle, q_0 de la porte **CX** est en superposition. Mais les règles physique (et de la porte **CX**) implique que si q_0 est dans l'état $|1\rangle$ alors q_3 est dans l'état $|1\rangle$. Nous avons donc deux états possible du système : $|00\rangle$ ou $|11\rangle$.

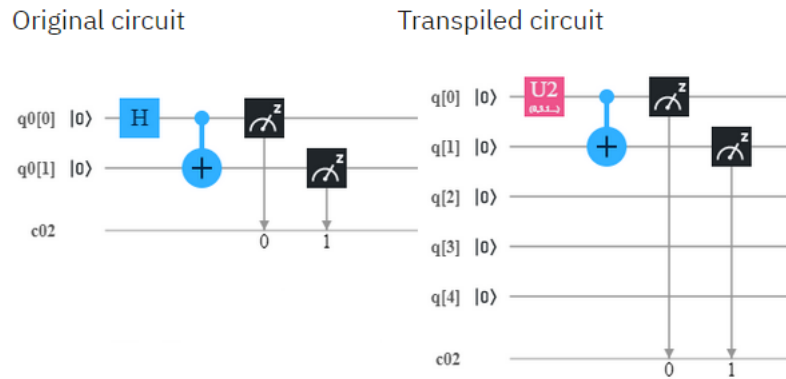


FIGURE 2.6 : Circuit avant et après transpilation

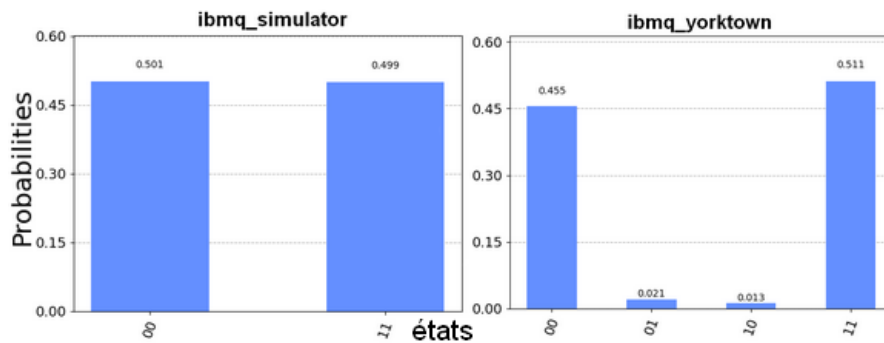


FIGURE 2.7 : Résultats pour l'exécution du circuit de la figure 2.5

La figure 2.7 illustre le second phénomène de la théorie quantique : l'intrication. Ici, le sort des états du qubit 0 et qubit 1 sont liés, le **CNOT** a invalidé l'état 10 et transformé celui ci en 11 (sur le simulateur). Enfin de nouveau, sur l'histogramme de droite de la figure 2.7, le bruit sur la machine réelle fait apparaître (très faiblement) les états "impossibles". Notons que les histogrammes traduisent déjà une faible perturbation

sur les machines réelles pour un circuit avec une porte binaire⁵.

Qiskit fournit également beaucoup d'informations sur les circuits créés et leurs exécutions, comme la taille du circuit qui est le nombre total d'opérations (portes quantiques) dans le circuit. Il y a également une deuxième notion qui est la profondeur, qui est calculée en regroupant des portes compatibles par couche.

⁵Par la suite, nous verrons qu'un simple enchaînement de plus de 47 portes rend les résultats illisibles car le taux d'erreur compris dans l'intervalle $[1, 47 \cdot 10^{-2} \dots 2, 14 \cdot 10^{-2}]^{47}$ dépasse déjà 50% de bruit.

Algorithmique de recherche

Il existe de plus en plus algorithmes quantiques, dont certains traitent de la recherche dans une base non structurée. C'est ce qui va nous intéresser en Optimisation Combinatoire. Citons l'algorithme de Grover [11], ainsi que d'autres algorithmes quantiques approchés tels que *Quantum Approximate Optimization Algorithm* [10] et *Variational Quantum Eigensolver* [17].

L'algorithme de Grover repose essentiellement sur deux principaux opérateurs : l'oracle et l'amplification. Tandis que QAOA et VQE reposent sur l'évolution d'un système quantique, tel que lorsque ce système atteint une énergie minimale, chaque coefficient du système est minimal et correspond à la solution approchée. Nous allons nous concentrer sur l'algorithme de Grover qui apporte la capacité de faire une recherche exacte. Les deux autres algorithmes permettant, qu'en a eux, des recherches fournissant des solutions approchées.

3.1 Grover

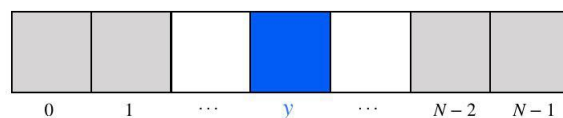


FIGURE 3.1 : Rechercher un élément dans un tableau

Pour trouver un élément dans un tableau non trié de longueur N , il n'existe pas d'algorithme séquentiel exact permettant de trouver cet élément en moins de $O(N)$ dans le pire des cas. Lov Grover proposa en 1996 un algorithme quantique de recherche d'éléments dans un ensemble non ordonné avec une complexité de \sqrt{N} , où $N = 2^n$ avec n le nombre de qubits nécessaires pour représenter la solution [11].

Cet algorithme nécessite $n + 1$ qubits, n qubits superposés pour représenter l'espace de recherche et un qubit supplémentaire pour effectuer les calculs. Enfin, nous répétons un certain nombre de fois (de l'ordre de racine N) les deux opérations suivantes :

- l'application d'une boîte noire (appelée aussi oracle), pour marquer la solution en inversant l'amplitude de l'état dans l'espace de recherche
- l'amplification d'amplitude. Amplifier la probabilité de l'état solution en inversant tout les coefficients des états autour de la moyenne des coefficients.

Présentons l'initialisation des états et les étapes principales de l'algorithme plus en détail. Nommons le registre de n qubits $|\Psi\rangle$ pour représenter l'espace de recherche et le qubit $|out\rangle$ pour stocker la valeur de l'oracle.

L'initialisation

Les qubits sont habituellement initialisés dans l'état $|0\rangle$ avant chaque opération. La première étape de l'algorithme consiste à superposer les états de l'espace de recherche en appliquant la porte de Hadamard, \mathbf{H} sur tout le registre $|\Psi\rangle$. Pour les besoins du calcul de l'oracle, il faut initialiser $|out\rangle$ dans l'état XH . Appliquons alors la porte \mathbf{X} puis la porte \mathbf{H} pour obtenir cet état.

Représentons cette opération de façon géométrique. Soit $|y\rangle$ l'état solution, notons que $|y\rangle$ et $|\Psi\rangle$ forment un plan complexe mais ils ne sont pas tout à fait orthogonaux car l'état solution $|y\rangle$ est également représenté dans la superposition des états de $|\Psi\rangle$. Intégrons $|x\rangle$ l'état orthogonal à $|y\rangle$ pour représenter notre plan complexe. Le registre $|\Psi\rangle$ est notre espace de recherche superposé et l'opération O correspond à l'oracle et G à Grover (Oracle puis amplification d'amplitude). Nous avons :

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right) |x\rangle + \sin\left(\frac{\theta}{2}\right) |y\rangle$$

et $\frac{\theta}{2}$ est l'angle défini par $\sin\left(\frac{\theta}{2}\right) = \frac{1}{\sqrt{N}}$, avec $N = 2^n$ car tous les états superposés sont équiprobables.

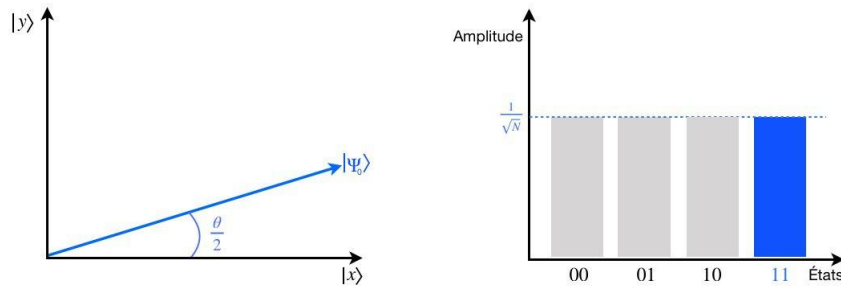


FIGURE 3.2 : *Initialisation*

Une fois que les états sont prêts, nous pouvons passer à l'itération de Grover en appliquant l'oracle puis l'amplitude d'amplification.

L'oracle

Le rôle de l'oracle va être de marquer les états solutions dans l'espace de recherche. Elle est définie en fonction du problème, elle va caractériser si un état correspond à

une solution ou non. L'oracle marque ces états en inversant la phase, c'est à dire en multipliant par -1 le coefficient α de l'état qui est une solution. Par exemple, posons l'espace de recherche suivant :

$$|\Psi\rangle = \alpha_0 |0\dots 0\rangle + \alpha_1 |0\dots 01\rangle + \dots + \alpha_N |1\dots 1\rangle$$

tel que la solution est $|1\dots 1\rangle$ alors l'application de l'oracle sur le registre $|\Psi\rangle$ va modifier le coefficient α_N de l'état solution qui deviendra $-\alpha_N$, et nous obtenons donc :

$$|\Psi\rangle = \alpha_0 |0\dots 0\rangle + \alpha_1 |0\dots 01\rangle + \dots + \alpha_{N-1} |1\dots 10\rangle - \alpha_N |1\dots 1\rangle$$

Ce marquage est effectué par l'intrication entre le qubit $|out\rangle$ initialisé avec les portes \mathbf{X} puis \mathbf{H} et $|\Psi\rangle$ tel que $|out\rangle$ est inversé si et seulement si l'état de l'espace de recherche correspond au critère de l'oracle. En d'autre terme, $|out\rangle$ sera dans l'état $|1\rangle$ si le registre $|\Psi\rangle$ est dans un état solution. Cela va avoir pour effet d'inverser la phase de l'état solution dans $|\Psi\rangle$ et de perturber la moyenne des α_i (en pointillé sur l'histogramme) par ce changement de phase.

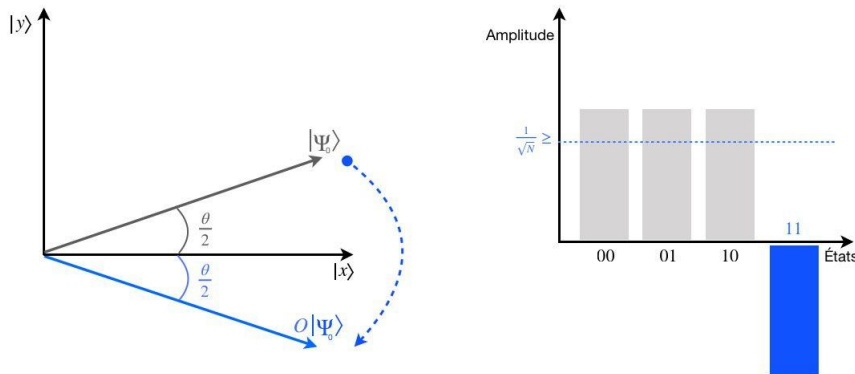


FIGURE 3.3 : *Oracle*

Traisons maintenant la dernière opération de cet algorithme, l'amplification d'amplitude qui a été découvert pour la première fois par Gilles Brassard et Peter Hoyer [3] puis redécouvert par Lov Grover [12].

Amplification d'amplitude

Jusqu'à présent, nous avons un registre $|\Psi\rangle$ d'états superposés tels que si on le mesure on a une probabilité uniforme de mesurer chaque état non solutions. L'opération d'amplification d'amplitude va permettre d'amplifier la phase des états cibles. Comme les états solutions sont marqués par l'oracle, cette opération va cibler les états solutions et augmenter leurs amplitudes tout en réduisant celles des autres états.

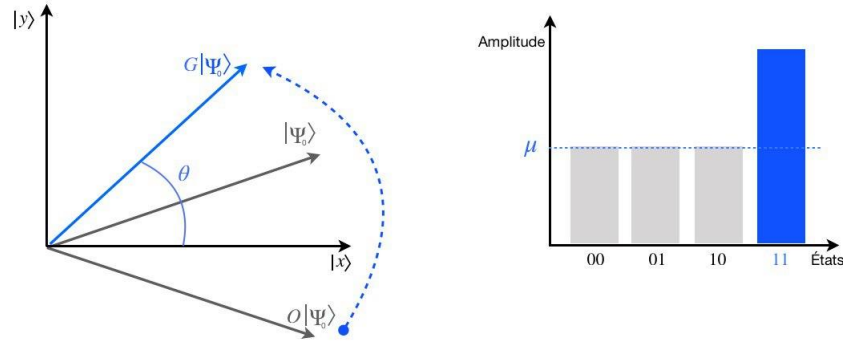


FIGURE 3.4 : Amplification d'amplitude

Sur cette figure, nous observons l'état $|\Psi\rangle$ inversé par l'oracle puis amplifié par l'amplification d'amplitude. Nous appliquons l'opération d'amplification :

$$A = 2|\Psi\rangle\langle\Psi| - 1$$

Cette opération rééquilibre les valeurs des états (voir l'histogramme de la figure 3.4). Une itération fait gagner un angle θ à l'unique état solution de l'oracle. Nous allons réitérer l'opération $G = OA|\Psi\rangle$, t fois.

Après t étapes, l'état $|\Psi\rangle$ devient l'état $|\Psi_t\rangle = G^t|\Psi\rangle$ et cet état va se rapprocher de l'état solution $|y\rangle$. Notons que si t est trop grand, nous allons dépasser l'état $|y\rangle$ et donc réduire la probabilité de la solution.

À chaque étape, l'angle est augmenté de $\theta = 2\arcsin\frac{1}{\sqrt{N}}$. L'objectif est de répéter l'algorithme de Grover afin d'approcher le plus possible l'état solution avec une probabilité définie par :

$$\sin^2\left(\theta\left(t + \frac{1}{2}\right)\right)$$

La plus haute probabilité est atteinte quand $t = \frac{\pi}{4}\sqrt{N}$. Il faut donc itérer ces opérations $0.78\sqrt{N}$. La complexité de l'algorithme de Grover est donc de l'ordre de $O(\sqrt{N})$.

Dans le cas où il y a plusieurs solutions, par exemple k , il faut répéter l'algorithme de Grover $\frac{\pi}{4}\sqrt{\left(\frac{N}{k}\right)}$ car l'oracle inverse les phases de tous les états solutions en même temps, ce qui augmente la valeur de θ et accélère la convergence. Mais k n'est pas toujours connu et on doit donc itérer Grover avec $k = 1$, puis $k = 2$, ... tant que k n'est pas correct. Pour tout k , le nombre total d'itérations est de :

$$\frac{\pi}{4}\sqrt{N}\left(1 + \frac{1}{\sqrt{2}} + \frac{1}{2} + \dots\right) = \frac{\pi}{4}\sqrt{N}(2 + \sqrt{2})$$

La complexité est toujours de l'ordre de $O(\sqrt{N})$.

Exemple

Présentons l'algorithme de Grover avec le petit exemple suivant :

- La solution est l'état : $|11\rangle$

- L'espace de recherche : $|\Psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$

Comme la somme des carrés des coefficients doit être égale à 1, les coefficients : $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{2^2}} = \frac{1}{2}$

Commençons par écrire dans notre circuit l'initialisation des états $|\Psi\rangle$ et $|out\rangle$:

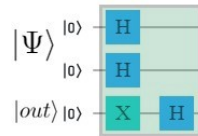


FIGURE 3.5 : Initialisation des états

L'oracle doit marquer la solution, ici $|11\rangle$ en inversant son amplitude : $\frac{1}{\sqrt{2}}$. Le circuit va effectuer cette opération avec la porte contrôlée **CCNOT**. Les qubits de contrôles seront ceux du registre de l'espace de recherche $|\Psi\rangle$ et le qubit cible sera $|out\rangle$. De fait, si les qubits de contrôles sont dans l'état $|1\rangle$ qui correspond à l'état $|11\rangle$ de $|\Psi\rangle$ alors nous inverserons $|out\rangle$ ce qui est attendu. Par la suite, il suffira simplement de changer la valeur de l'oracle pour représenter un autre révélateur.

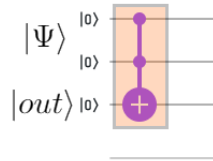


FIGURE 3.6 : Portion de circuit représentant l'oracle

Nous ajoutons l'opération d'amplification d'amplitude après l'oracle :

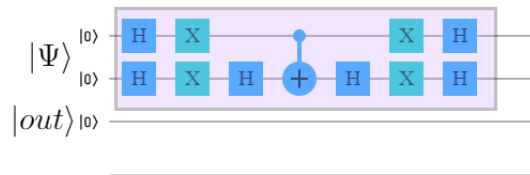


FIGURE 3.7 : Portion de circuit représentant l'amplification d'amplitude

Comme nous avons une solution, nous sommes dans le cas $k = 1$ et nous devons donc répéter Grover $\frac{\pi}{4}\sqrt{2^2} = \frac{\pi}{2}$ fois. De façon géométrique, on sait que $\frac{\theta_t}{2}$ doit être proche de $\frac{\pi}{2}$. Comme $\sin\frac{\theta_0}{2} = \frac{1}{\sqrt{N}}$, on a $\theta_0 = 2\arcsin\frac{1}{\sqrt{N}}$. Donc $\theta_0 \approx 1.07$ radian. Si on répète Grover $\frac{\pi}{2}$ fois, alors on va augmenter l'angle θ tel que $\theta_t = \frac{\pi}{2}2\arcsin\frac{1}{\sqrt{N}} = \frac{\pi}{2}$. On va donc répéter Grover 1 fois.

Voici le circuit final :

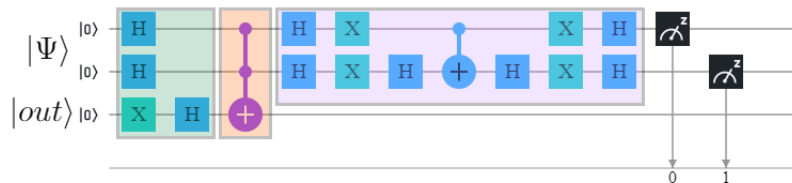


FIGURE 3.8 : Le circuit de Grover complet

Le résultat obtenu en exécutant 1024 tirages du circuit sur le simulateur et la machine quantique *ibmq_yorktown* :

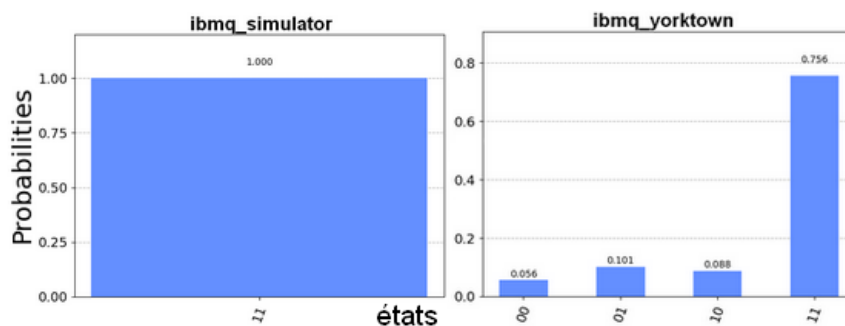


FIGURE 3.9 : Résultat du circuit en figure 3.8

Comme sur les exécutions présentées en 2.7 et 2.5 nous remarquons également des résultats bruités (00, 01 et 10).

Ce qu'il faut retenir, c'est que l'algorithme de Grover se fait en 3 parties, une initialisation superposant tous les états possibles du système, un oracle décrivant ce que l'on cherche afin de marquer la solution et un révélateur isolant la solution de tous les états superposés. Désormais, oublions la physique¹. Nous allons voir comment cet algorithme générique va être très utile.

3.2 3 SAT

SAT, pour SATisfiabilité est un problème très important dans l'informatique, notamment dans la théorie de la complexité. Le problème 3-SAT a été le premier des problèmes démontré NP-complets [5]. Nous allons utiliser l'algorithme de Grover pour résoudre le problème 3-SAT. Avec l'aide de quelques qubits, représentant les littéraux, nous allons superposer tous les états possibles de l'espace de recherche. Ensuite il "suffit" juste d'encoder SAT dans la partie "Oracle". Enfin La partie amplification d'amplitude servira à mettre en évidence la solution.

¹Nous aurions pu commencer le rapport à cet endroit en nommant seulement les trois étapes de l'algorithme de Grover mais, nous avons jugé utile de les présenter.

SAT est un problème qui admet déjà un circuit dans la librairie de qiskit. La documentation ne présente pas d'algorithme de façon explicite, mais fournit en référence un article écrit par Giacomo Nannicini [14], qui traite du problème **exactly one 3 SAT** pour des formules de trois variables et de trois clauses. Cet article contient une partie algorithmique détaillée et du code python. Cet article est clairement orienté pour les informaticiens, il est d'ailleurs intitulé : "Quantum computing without physics". Comme notre démarche est d'essayer de comprendre et d'expliquer comment programmer un circuit quantique et comment fonctionne ces circuits d'un point de vue d'informaticiens, nous avons réalisé notre propre circuit pour résoudre le problème 3 SAT en nous inspirant de l'article *Quantum computing without physics*.

Dans un premier temps, nous allons expliquer comment résoudre des instances de 3 SAT et une généralisation du circuit pour résoudre des instances de SAT. Les explications seront accompagnées d'exemples et d'algorithmes. À la fin de ce chapitre, il est présenté différentes exécutions sur simulateur et machines réelles ainsi qu'une évaluation du coût en espace.

L'oracle

Pour résoudre une instance du problème **SAT** nous devons pouvoir calculer la valeur d'une formule sous forme normale conjonctive. Nous avons donc besoin des opérateurs : **NON**, **OU** et **ET**. Dans le chapitre 2, nous avons vu qu'il était possible de décrire toutes les fonctions logiques car nous avons la possibilité de reproduire les opérateurs **ET** logique avec la porte **CCNOT** et le **NON** avec la porte **X**. Le **OU** peut être reproduit avec le **NON** et le **ET** de la façon suivante :

$$x_0 \vee x_1 \equiv \neg(\neg x_0 \wedge \neg x_1)$$

Les étapes principales de l'oracle vont être les suivantes : dans un premier temps nous devons encoder les clauses. Ensuite, nous allons calculer la valeur de chaque clause pour finalement calculer la valeur de la formule.

Ainsi, le registre de qubits $|\Psi\rangle$ représente l'espace de recherche correspondant aux variables. Le qubit représentant la valeur de la formule est le qubit $|out\rangle$ et il sera dans l'état $|1\rangle$ si l'assignation des variables est valide. Il y a également m qubits auxiliaires $|aux_i\rangle$, avec m le nombre de clauses, pour stocker les valeurs des clauses et qui seront également dans l'état $|1\rangle$ si la clause est satisfaite.

Exemple

Dans cet exemple, nous allons présenter comment écrire une clause, calculer sa valeur et comment calculer la valeur finale d'une formule dans un circuit quantique.

Soit la clause $C_0 = x_0 \vee x_1 \vee \neg x_2$ suivante. Voici le circuit qui encode cette clause :



FIGURE 3.10 : Représentation de la clause c_0

La porte **X** permettant d'inverser l'état d'un qubit (afin de travailler avec sa négation) est appliquée uniquement sur x_2 .

Une fois la clause C_0 encodée, il faut calculer sa valeur et la stocker dans $|aux_0\rangle$ en vue de l'évaluation finale en appliquant un **OU** sur l'état $|\Psi\rangle$, les variables.

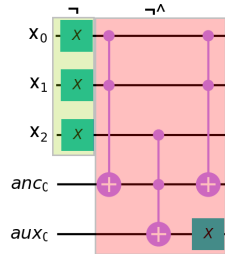


FIGURE 3.11 : Calcul de la clause c_0

Il ne faut pas oublier de réinitialiser les qubits pour la prochaine clause C_1 sinon le calcul va être faussé pour $|aux_1\rangle$. Nous remarquons finalement que pour les littéraux négatifs, nous allons appliquer 2 portes **X** et 1 porte **X** pour les littéraux positifs. Nous pouvons donc réécrire le calcul d'une clause en n'appliquant seulement la porte **X** que sur les littéraux positifs. Par exemple, la clause $C_0 = x_0 \vee x_1 \vee \neg x_2$ devient $\neg(\neg x_0 \wedge \neg x_1 \wedge x_2)$:

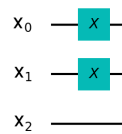


FIGURE 3.12 : Représentation de la clause $\neg c_0$

Une fois toutes les valeurs des clauses stockées dans les qubits auxiliaires, calculons la valeur de la formule dans out_0 , toujours avec un triple **CCX** et un qubit d'ancilla (anc_0).

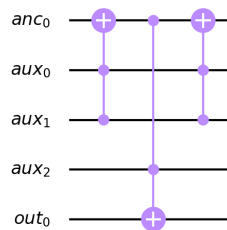


FIGURE 3.13 : Calcul de la formule

Présentons à présent une itération complète de la recherche de Grover pour le problème SAT en prenant comme exemple la formule suivante : $f = (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2) \wedge (\neg x_0 \vee x_1 \vee x_2)$

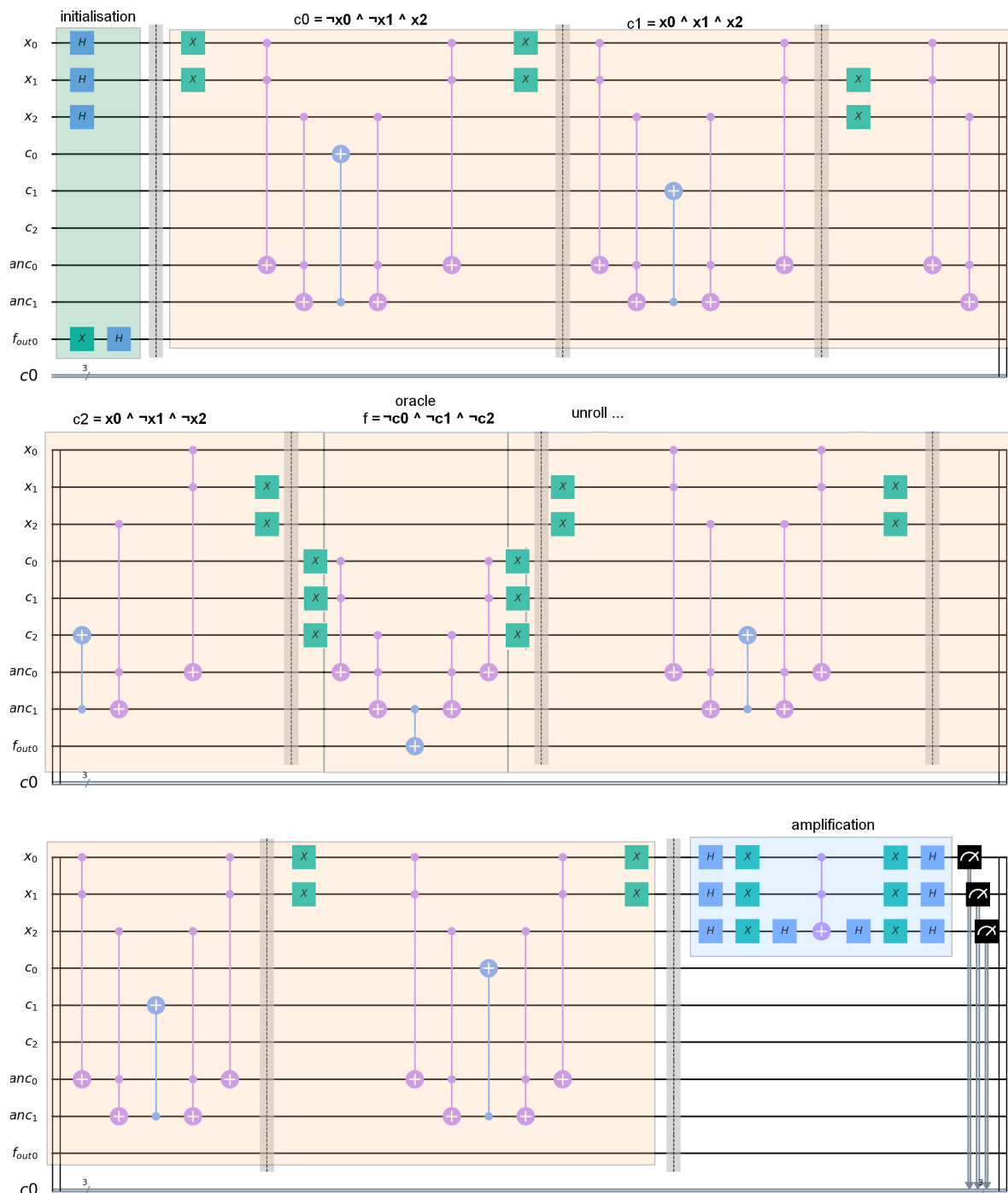


FIGURE 3.14 : Circuit 3SAT pour la formule f

Remarquons les trois étapes de l'algorithme de Grover, l'initialisation, l'oracle décomposé en deux parties (lecture des données et calcul) et la troisième étape, l'amplification.

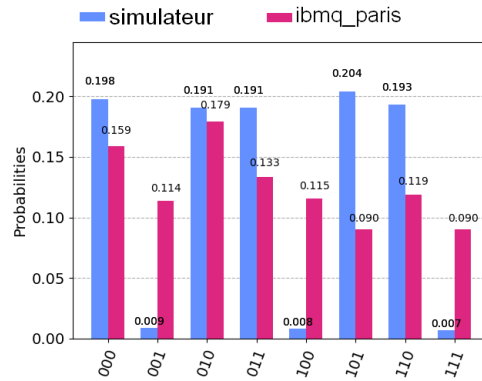


FIGURE 3.15 : Résultats du circuit en figure 3.14

Les 5 solutions apparaissent clairement sur l’histogramme du simulateur². Pour rappel, les chaînes binaires correspondant aux états se lisent de droite à gauche. Par exemple, l’état 110 correspond à l’affectation $x_2 = 1$, $x_1 = 1$ et $x_0 = 0$.

Cette figure nous montre la non-fiabilité des machines actuelles, ce qui ne permet pas encore hélas de pouvoir résoudre la plus petite instance de 3SAT à 3 clauses. Ici, le circuit exécuté contient 362 portes binaires ce qui correspond à un taux de réussite de 0.0006% et un taux d’échec de 99,9994% (voir calcul effectué en fin de section 2).

Coût en espace

Cette modélisation a un coût en nombre de qubits. Comme nous sommes actuellement limité par les machines et simulateurs disponibles, nous avons étudiés le coût en espace et en opérations de notre circuit non optimisé. Cela permet d’avoir un regard sur la taille des formules que l’on peut exécuter aujourd’hui et celle que nous pourrions exécuter plus tard.

Pour une formule **SAT** avec n variables et m clauses l’oracle 1 présenté utilise :

- n qubits pour représenter l’espace de recherche $|\Psi\rangle$
- m qubits auxiliaires pour stocker les valeurs des clauses
- $m - 2$ qubits d’ancillas
- 1 qubit pour la valeur de l’oracle

Au total, nous avons donc besoin de $n + 2m - 1$ qubits pour résoudre formule **SAT**.

Pour l’instant, les machines qui ont le plus grand nombre de qubits sont *ibmq_rochester* d’IBM avec 53 qubits et la nouvelle puce quantique Sycamore de 54 qubits de Google. Sans prendre en compte les problèmes liés aux erreurs de calculs pendant les exécutions, il n’est pour l’instant pas possible d’exécuter de gros exemples de formules. Comme vu sur la figure précédente, même le plus simple des 3SAT (3 variables, 3 clauses) ne peut être que simulé et ne fonctionne pas sur machine.

²En fait l’algorithme nous fournit non pas une solution mais toutes les solutions de la formule quelle que soit l’instance (quel que soit le nombre de solutions). Il est donc possible de faire un lien avec #3-SAT qui retourne le nombre de solution pour une instance de 3-SAT.

3.3 Exécutions

Dans un premier temps, nous avons exécuté des formules possibles 3 SAT avec 3 variables avec un nombre de clauses allant de 3 à 8. Le nombre de clauses satisfiables possibles avec 3 variables est 8, donc la plus grande formule que nous allons pouvoir construire sera de taille 8. Au total, nous allons avoir 219 formules à exécuter.

ce qui équivaut à

$$\sum_{i=3}^8 \binom{8}{i} = \sum_{i=1}^8 \binom{8}{i} - \sum_{i=1}^2 \binom{8}{i}$$

Comme :

$$(a + b)^k = \sum_{i=0}^k \binom{k}{i} a^{k-i} b^i$$

On a pour $a = b = 1$:

$$(1 + 1)^k = \sum_{i=0}^k \binom{k}{i} 1^{k-i} 1^i = \sum_{i=0}^k \binom{k}{i}$$

Donc,

$$\sum_{i=3}^8 \binom{8}{i} = 2^8 - \left(\binom{8}{1} + \binom{8}{2} \right) = 256 - (8 + 28) = 220$$

En éliminant le cas où on ne prend rien ($k=0$), on obtient 219 formules.

Le graphique, ci-dessous, présente la durée d'exécutions en secondes de chaque circuit sur le simulateur et la machine *ibmq_paris* (axe secondaire). Le simulateur n'est pas celui d'IBM présent sur le *cloud* mais mon ordinateur personnel. Nous avons fait ce choix car nous nous sommes rendu compte que les temps d'exécutions pouvaient être très différent pour un même circuit. En effet, IBM nous ont confirmé qu'il pouvait y avoir des fluctuations de temps pendant l'accès au système de fichiers car il dépend de ce qui s'exécute.

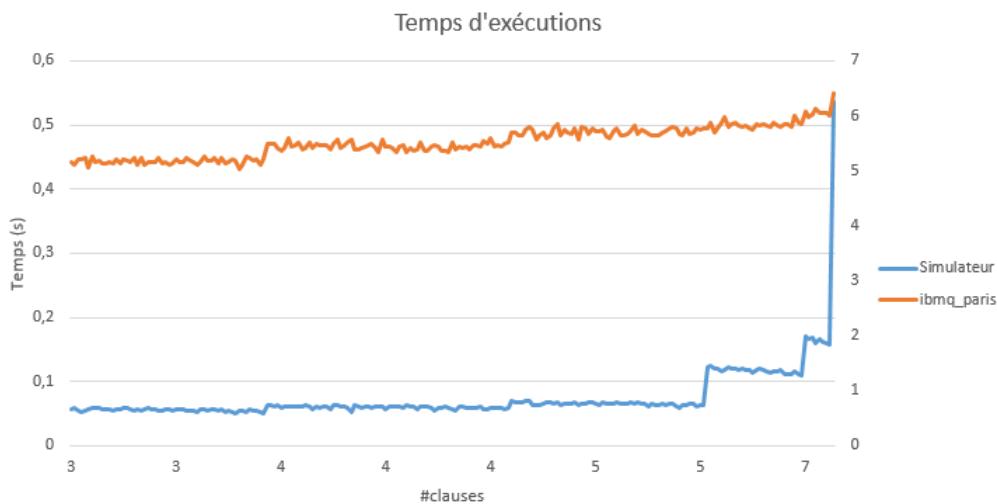


FIGURE 3.16 : Résultats sur simulateur et sur *ibmq_paris*

Le temps d'exécution sur la machine *ibmq_paris* (axe secondaire) peut paraître beaucoup plus important que ceux du simulateur. En fait, ce temps comprend toute la procédure (calibration, écriture du circuit, exécution, lecture des résultats). En réalité le temps de cohérence des qubits est de l'ordre de quelques nanosecondes. Ce qui est important ici est d'observer la variation du temps. Nous voyons clairement la progression exponentielle du nombre d'états superposés des qubits, énumérés séquentiellement dans le simulateur sur une machine classique, et évalués simultanément sur une machine quantique.

Sur la figure, ci-dessous, nous présentons la taille qui est le nombre d'opérations dans les circuits. Les données concernent les tailles des circuits transpilés sur la machines *ibmq_paris*.

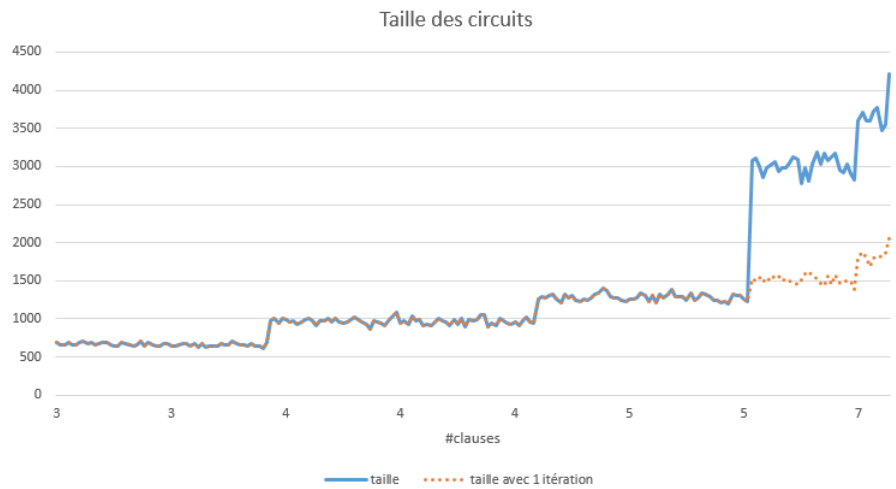


FIGURE 3.17 : Tailles des circuits sur *ibmq_paris*

Les premières formules sont de taille 3 clauses et la dernière (219 ième) est de taille 8. Nous remarquons plusieurs paliers en fonction du nombre de clauses et quelques variations au sein de ces paliers en fonction du nombre de portes \mathbf{X} . Comme expliqué précédemment, il faut ajouter 2 qubits auxiliaires par clause, ce qui augmente la taille en espace du circuit. Il faut aussi prendre en compte le nombre d'itérations de Grover. En effet, nous avons vu en 3.1 que le nombre d'itérations dépendait du nombre de solutions. Nous savons que chaque assignation invalidant une clause rend la formule fausse et que ces assignations ne constituent pas une solution. Donc, plus la formule a de clauses (de contraintes) moins il y aura de solutions. Ceci va avoir comme conséquence d'augmenter le nombre d'itérations et donc la taille de notre circuit. La courbe en pointillé orange correspond à la taille des circuits avec une seule itérations de Grover. Pour information, la courbe représentant la profondeur des circuits suit la même tendance.

3.4 Recherche dans un tableau

Nous allons voir dans un premier temps une méthode pour encoder un tableau de taille n avec deux registres, l'un de taille $O(\log(n))$ pour les index, l'autre de taille $O(\log_2(D))$,

D le majorant des valeurs dans le tableau.

Nous allons supposer que le tableau ne contient pas plusieurs fois la même valeur. L'algorithme d'encodage est inspiré de [4].

Idée générale

Un tableau est une relation entre une valeur et son index. Dans la sous-section précédente, nous avons vu une façon d'encoder des entiers, nous pouvons donc avoir un registre encodant un index et un autre la valeur.

L'idée va être d'intriquer le registre codant un index avec un registre codant sa valeur dans le but de créer cette relation. Comme il est possible superposer le registre d'index, il n'est pas nécessaire d'avoir un registre pour chaque index.

Si nous utilisons la superposition d'un des deux registres, il est possible d'encoder un tableau en $2\log_2 k$ avec k la valeur maximale dans le tableau.

Algorithme

Algorithme 2 : Encodeur

entrées : circuit, $|i\rangle$, pour les indices, $|v\rangle$ pour $t[i]$ et $|ancillas\rangle$, pour le registre d'ancillas, t le tableau à encoder

sortie : circuit : Circuit quantique

```

1 circuit.h( $|i\rangle$ )
2 pour chaque  $k \in [0, size(i)]$  faire
3   encode(circuit, k,  $|i\rangle$ )
4    $e \leftarrow \text{bitfield}(t[k])$ 
5   pour chaque  $l \in [0, size(v)]$  faire
6     si  $e[l] == 1$  alors
7       |  $\text{n\_cnot}(\text{circuit}, |i\rangle, |ancilla\rangle, |v\rangle[l])$ 
8     fin
9   fin
10  encode(circuit, k,  $|i\rangle$ )
11 fin
12 retourner circuit

```

1. On place l'état des index en superposition
2. Pour chaque élément dans le tableau,
 - a) On encode l'index correspondant sur le registre $|i\rangle$
 - b) On convertit la k-ième valeur du tableau en binaire
 - c) Pour chaque bit positif, on applique un **C_NOT** entre les index et le qubit devant valoir $|1\rangle$
3. On *unroll* le circuit sur $|i\rangle$

Étudions maintenant la recherche dans un tableau. L'oracle que nous allons décrire fournira l'indice dans le tableau de l'élément e recherché.

Comme le registre d'indices est intriqué avec le registre de valeurs, il faut que l'oracle soit activé seulement si le registre de valeurs représente e (en binaire). Nous allons utiliser la porte **N_CNOT** sur le registre de valeurs et le qubit de l'oracle. En l'état, l'oracle serait actif seulement si tous les qubits du registre de valeurs sont dans l'état $|1\rangle$. Dans le cas où $e = 2^n - 1$, avec n la taille du registre de valeurs, le résultat serait correct par intrication index/valeurs. Pour le cas général, il faut appliquer la porte **X** sur les qubits dans l'état $|0\rangle$ dans la représentation binaire de e , ce qui mettra tout les qubits de contrôle (le registre de valeurs) de la **N_CNOT** dans l'état $|1\rangle$. Par exemple, si $e = 3$ et $n = 3$, la représentation de 3 est 011. Nous devons donc faire en sorte que l'oracle soit dans l'état $|1\rangle$ si et seulement si le registre de valeurs est dans l'état $|011\rangle$. Si nous ajoutons la porte **X** sur le qubit de poids fort, nous obtenons l'état $|111\rangle$ seulement si le registre de valeurs est dans l'état $|011\rangle$, ce qui activera la **N_CNOT** et l'oracle également.

Un exemple avec le tableau $T = [0, 7, 3, 5]$ et $e = 3$, l'élément à rechercher dans T .

L'oracle va d'abord procéder à la phase d'encodage. Regardons l'encodage de 7 $[1,1,1]$ qui s'active sur les trois qubits de valeur v_0, v_1 et v_2 quand l'index vaut $[1,0]$ sur les qubits d'indice i_0, i_1 (il est inversé avec un **X**) :

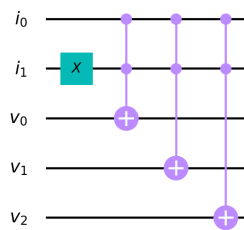


FIGURE 3.18 : Encodage de 7 $([1,1,1])$

Après avoir encodé toutes les valeurs du tableau, il faut préparer la valeur à rechercher, e , sur le registre des valeurs $|v\rangle$.

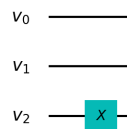
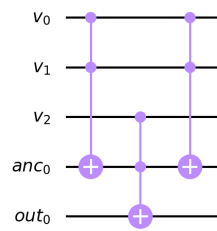


FIGURE 3.19 : Recherche de 3 $[1,1,0]$ sur le tableau $[v_0, v_1, v_2]$

Comme nous recherchons la valeur $e = 3$, il faut que la registre $|v\rangle$ corresponde à la chaîne binaire $|011\rangle$ avant l'application des portes **X** nécessaire à l'activation de l'oracle. Nous plaçons donc la porte **X** sur le qubit v_2 .

Enfin nous intriquons le registre de valeurs $|v\rangle$ avec $|out\rangle$:

FIGURE 3.20 : N_CNOT pour activer l'oracle

Ensuite, il ne faut pas oublier l'étape de *unroll* sur le circuit. Présentons le circuit récapitulatif de la recherche de l'élément $e = 3$ dans le tableau $T = [0, 7, 3, 5]$:

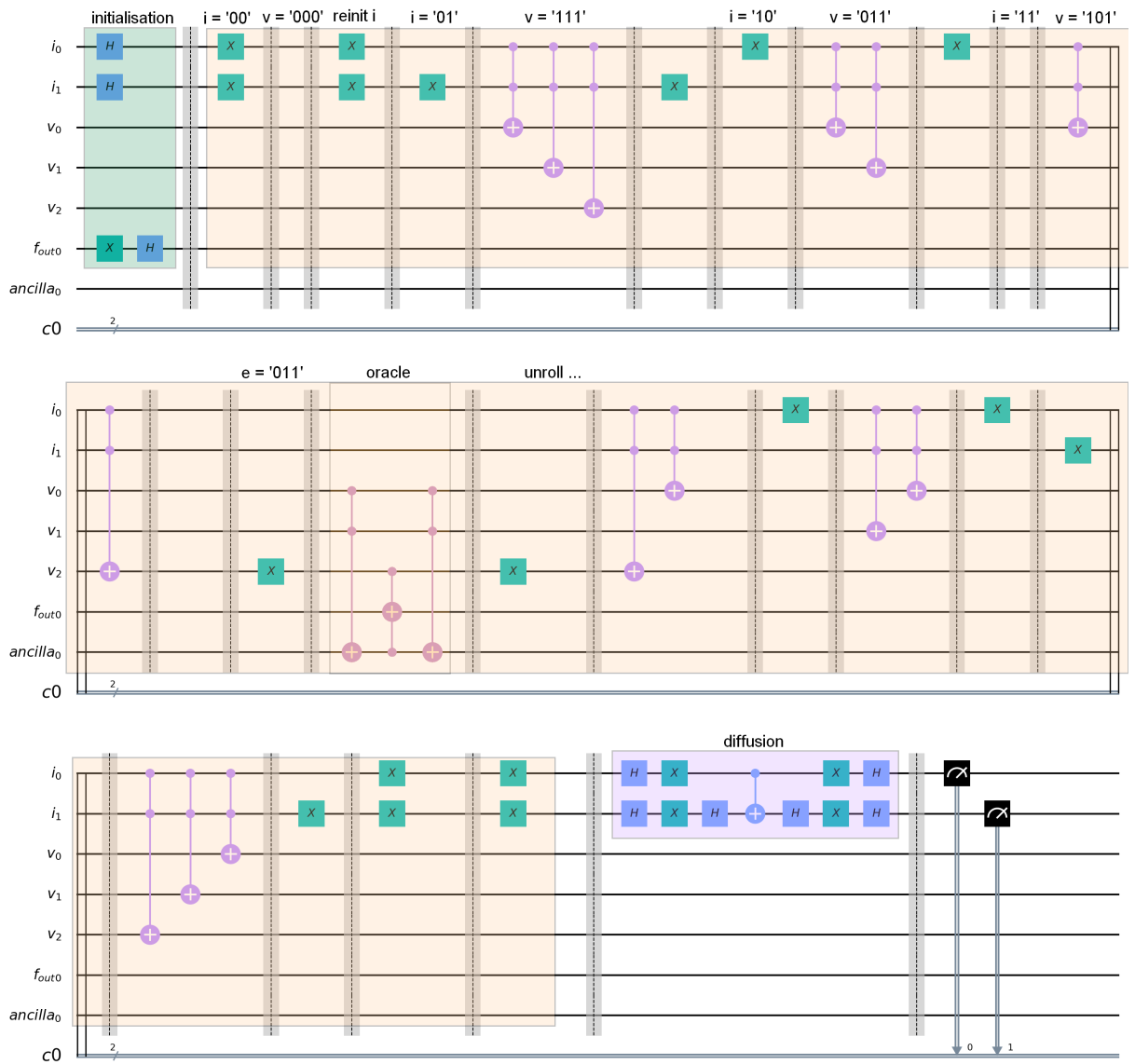


FIGURE 3.21 : Le circuit complet

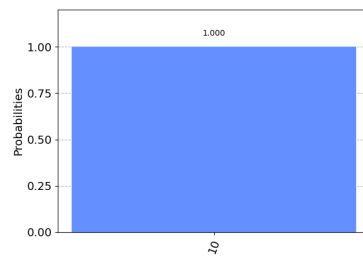
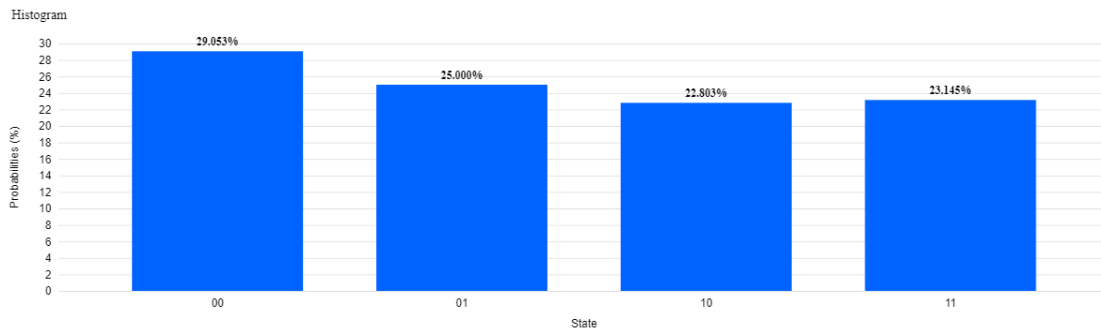


FIGURE 3.22 : Résultat sur simulateur (index 10)

Sur la figure, ci-dessus, nous remarquons sur l'histogramme le résultat, la chaîne : 10, obtenu à partir d'une exécution sur simulateur du circuit présenté en figure 3.20. Remarquons que 3 est en position 2 représenté par la chaîne binaire 10, et donc le résultat est correct.

FIGURE 3.23 : Résultat sur la machine *ibmq_paris*

Sur le graphique 3.23, nous voyons que la machine *ibmq_paris* n'est pas encore assez fiable pour trouver un élément dans un tableau de taille 4. En effet, comme déjà expliqué, cela est dû au taux d'erreur des machines qui pour ce programme une fois transpilé et compilé sur la machine, possède 185 portes unitaires et 325 portes binaires ce qui fait un taux de réussite de 0.14%.

Conclusion

Dans ce chapitre, nous avons traité le problème SAT, pièce maîtresse de la théorie de la complexité donnant accès par réduction à une complexité en $O(\sqrt{N})$. Est également introduit comment représenter les entiers, ainsi qu'un tableau d'entier et sa manipulation sur une machine quantique. Tout est prêt à présent pour traiter des problèmes combinatoires entiers.

Optimisation Combinatoire

La minimisation et la maximisation sont des problèmes classiques en informatique, plus particulièrement en optimisation combinatoire. En effet, minimiser ou maximiser une fonction permet de résoudre le problème d'optimisation modélisé par cette même fonction (couverture de sommets, voyageur de commerce, coloration).

Précédemment, nous avons vu qu'avec le paradigme quantique nous pouvons superposer un registre de qubits afin de travailler sur toutes les valeurs possibles. Ce registre superposé va être notre ensemble de solutions. Par exemple pour la minimisation d'une fonction entière, l'ensemble de solutions sera un registre d'entiers. Également, nous avons vu qu'avec l'algorithme de Grover et un oracle dédié, nous pouvons extraire des états solutions parmi un ensemble d'états superposés.

La stratégie pour minimiser une fonction va être la suivante ; superposer l'espace de recherche sur un registre de qubits, construire un oracle calculant le minimum parmi les valeurs superposées et utiliser l'opérateur de diffusion de Grover pour extraire la ou les solutions.

4.1 Recherche du minimum

Notre objectif est de trouver l'élément minimum dans un tableau/ensemble d'entiers non trié. Nous avons besoin d'un circuit intermédiaire essentiel : un comparateur d'entiers. Avec ce circuit, nous allons pouvoir chercher les états inférieurs à une valeur k de façon successive jusqu'au minimum.

Présentons dans un premier temps le circuit "inférieur" suivi d'un exemple avec un tableau d'entiers non trié. Enfin, nous allons pouvoir utiliser ce circuit comme sous-routine de l'algorithme de Grover.

Comparateur d'entier

Le comparateur utilisé est celui présenté dans [16]. Le circuit admet deux qubits pour le résultat. En fait, le premier qubit correspond à la comparaison : $<$ et le second à la comparaison $>$. Si les deux qubits ont pour valeur l'état $|0\rangle$, cela veut dire que l'entier comparé n'est ni inférieur ni supérieur mais égal au second. Ici nous allons seulement présenter le cas inférieur.

Le principe de ce circuit est le suivant : nous allons comparer, qubit à qubit, les états en commençant par le *MSQ*, le qubit de poids le plus fort. Si l'un des deux qubits est plus grand (resp. plus petit) alors le résultat est déjà fini et cette information est transmise par les qubits d'*ancillas* de comparaisons¹ pour les prochaines opérations. Si ce n'est pas le cas (égalité par exemple), l'information va également être transmise par les *ancillas* de comparaisons pour les prochaines évaluations. En effet, nous remarquons qu'à chaque nouvelle évaluation, nous faisons un **N_CNOT** entre toutes les *ancillas* de comparaisons déjà traitées et les qubits en cours d'évaluation. Il suffit donc qu'une des *ancillas* de comparaisons soit dans l'état $|0\rangle$ pour que la porte **N_CNOT** ne s'applique pas, ce qui ne changera pas le résultat courant.

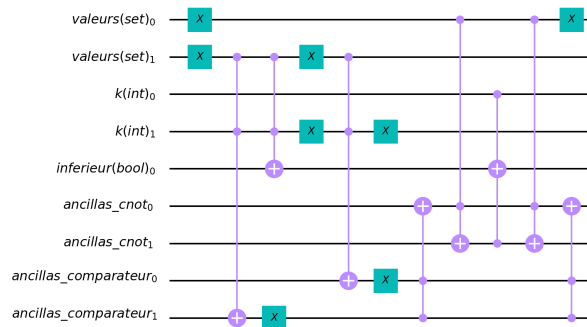


FIGURE 4.1 : Circuit qui compare si la chaînes binaires "valeurs" < "k".

Le *MSQ* est le qubit en position 1 pour les registres $|valeurs\rangle$ et $|k\rangle$. La première étape consiste à comparer le qubit $|valeurs_1\rangle$ avec $|k_1\rangle$. Si $|valeurs_1\rangle$ est dans l'état $|0\rangle$ et $|k_1\rangle$ dans l'état $|1\rangle$ alors $|valeurs\rangle$ est inférieur à $|k\rangle$. Ce qui est bien retranscrit par la première **CCX**. Le **X** sur $|valeurs_1\rangle$ change son état de $|0\rangle$ à $|1\rangle$, comme $|k_1\rangle$ est dans l'état $|1\rangle$ la première **CCX** est active et la seconde aussi. L'état du qubit $|inferieur\rangle$ passe dans l'état $|1\rangle$ et le qubit $|ancillas_comparateur_1\rangle$ dans l'état $|1\rangle$ puis $|0\rangle$ avec la porte **X**. Comme dit précédemment, la suite du calcul dépend de ce qui est déjà calculé, et à ce stade nous savons déjà que $|valeurs\rangle$ est inférieur à $|k\rangle$, donc nous ne devons pas continuer le calcul. Il s'agit du rôle de $|ancillas_comparateur_1\rangle$ qui dans l'état $|0\rangle$ d'empêcher toute autre application des **CCX**. Dans le cas contraire, les **CCX** n'auraient ni modifié $|ancillas_comparateur_1\rangle$ qui serait resté dans l'état $|0\rangle$ puis $|1\rangle$ par l'application de la porte **X** pour ne pas bloquer les prochains calculs, et ni modifié l'état du qubit $|inferieur\rangle$. Notons qu'il y a une deuxième comparaison pouvant bloquer la suite du calcul, c'est à dire le cas où $|k\rangle$ est supérieur à $|valeurs\rangle$. C'est le rôle du qubit $|ancillas_comparateur_0\rangle$. Observons que si la porte **CCX** n'est pas active, alors la porte **X** qui suit inversement son état qui devient $|1\rangle$, ce qui ne bloque pas les prochains calculs. Remarquons également que le seul cas où la **CCX** change l'état de $|ancillas_comparateur_0\rangle$ est quand $|k_1\rangle$ est dans l'état $|0\rangle$ (puis inversé par le **X**) et $|valeurs_1\rangle$ dans l'état $|1\rangle$. Nous avons bien dans ce cas, $|valeurs\rangle$ supérieur à $|k\rangle$.

¹le registre $|ancillas_comparateur\rangle$.

Exemple illustratif

Voici un exemple avec la structure de tableau présenté en section 3.4. Prenons les paramètres $n = 3$ et $k = 1$ et $T = [2, 1, 3, 0]$. Pour encoder le tableau T , nous allons utiliser l'algorithme d'encodage présenté précédemment.

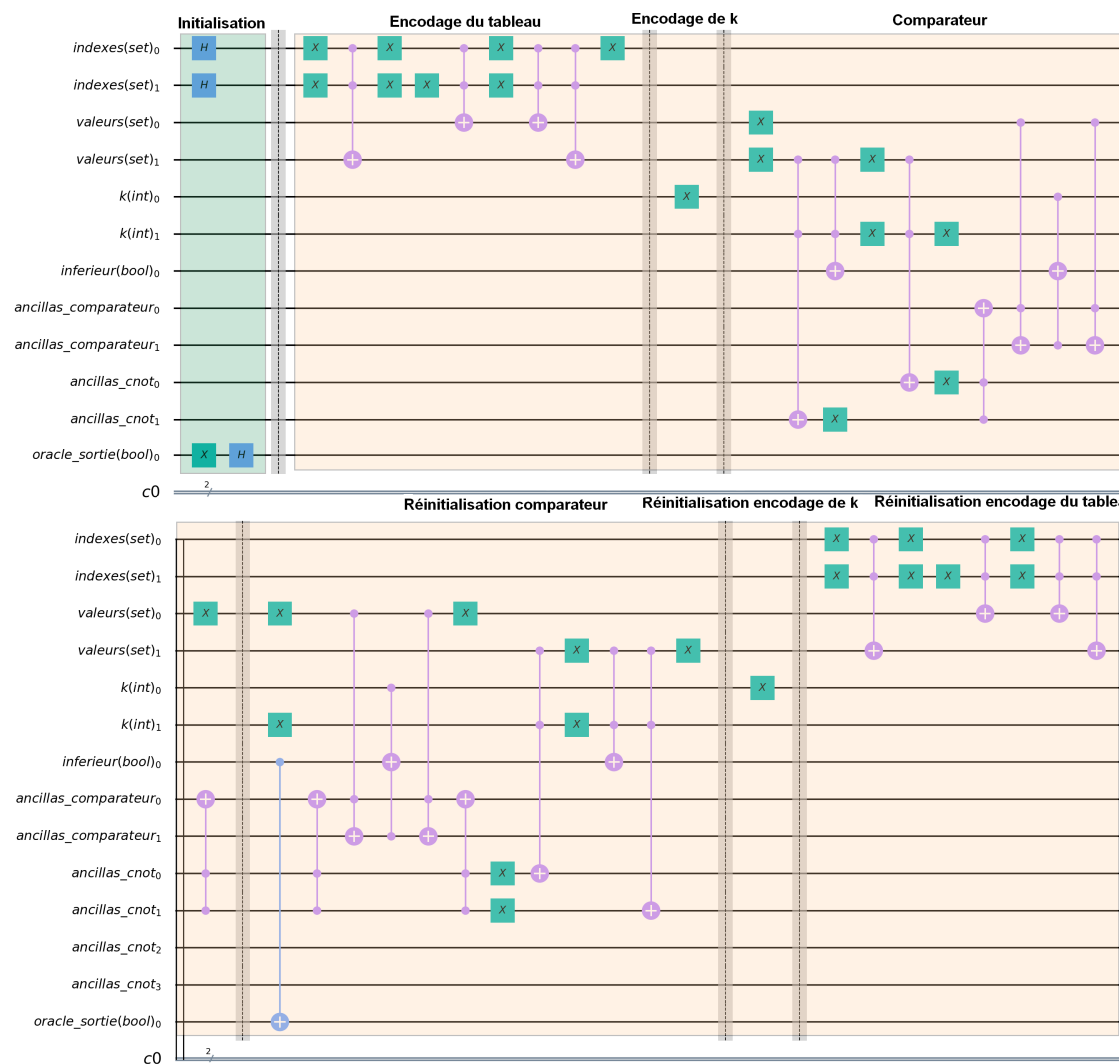


FIGURE 4.2 : Le circuit complet pour la recherche < 1

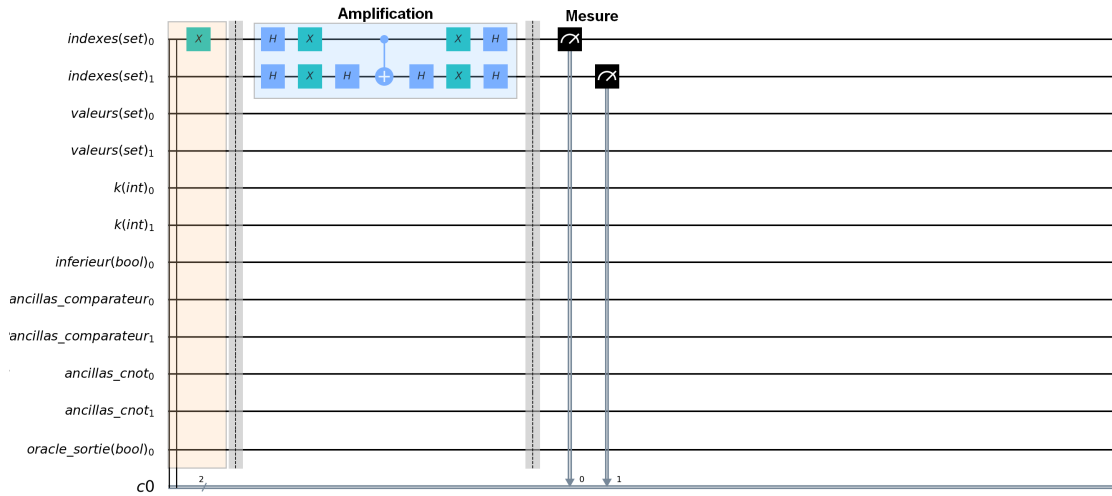


FIGURE 4.3 : Suite du circuit en figure 4.2

Le résultat est l'indice de l'élément 0 dans T , 3 représenté par l'état $|11\rangle$:

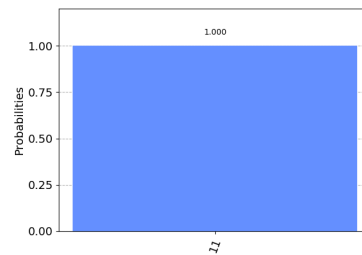


FIGURE 4.4 : Le résultat est la chaîne binaire 11

Complexité

L'algorithme *inférieur-k* fonctionne sur la structure de tableau et admet une complexité de $O(\sqrt{\frac{N}{M}})$ avec M le nombre de solutions et N , l'espace de recherche (2^n). À noter que cet algorithme trouve en $O(\sqrt{N})$ cette valeur dans un tableau non trié, non structuré, dans le pire cas. Il s'agit déjà d'un algorithme innovant par rapport à ce que nous connaissons classiquement en $O(N)$.

Revenons maintenant au problème principale : la recherche du minimum dans un tableau. Nous allons de nouveau être capable de trouver ce minimum en $O(\sqrt{N})$.

Recherche d'un élément minimum dans un tableau non trié

Obtenir le minimum d'un tableau non trié revient finalement à appliquer plusieurs fois l'algorithme *inférieur-k* vu précédemment. Tant qu'il existe un élément e' dans le tableau plus petit que l'élément e précédemment trouvé, il nous faut rappeler l'algorithme

inférieur- k . Dès qu'il n'y a plus de solution (aucun états satisfaisant l'oracle), la distribution en sortie devient équiprobable et il faut retourner l'avant dernier résultat.

Il existe un algorithme probabiliste présenté dans [9] avec une complexité $O(22.5\sqrt{N}1.4lg^2(N))$ avec une probabilité de 0.5 de succès. Si l'algorithme est répété c fois, la probabilité est de $1 - \frac{1}{2^c}$.

Présentons un algorithme dichotomique hybride pour la recherche du minimum dans une structure de type tableau. Nous allons utiliser des circuits comme sous procédure comme $Encode(qc, |k\rangle, k)$ qui écrit l'entier k sur le registre $|k\rangle^2$.

Algorithme 3 : Min-dicho

entrées : circuit, $|i\rangle$, pour les indexes, $|v\rangle$ pour $t[i]$ et $|ancillas\rangle$, pour le registre d'ancillas, $|f_out\rangle$, pour la valeur de l'oracle, $|k\rangle$, le registre de la valeur à comparer, t le tableau

sortie : k : Circuit quantique

- 1 initialisation_grover(qc, $|i\rangle$, $|f_out\rangle$)
- 2 $min \leftarrow -1$
- 3 $debut \leftarrow 0$
- 4 $fin \leftarrow 2^n$
- 5 $k \leftarrow \frac{debut+fin}{2}$
- 6 **tant que** $debut < fin$ **faire**
- 7 **pour chaque** $j \in [0, \frac{\pi}{4}\sqrt{\frac{N}{M}}]$ **faire**
- 8 Partie s'exécutant sur la machine quantique
- 9 Encodeur(qc, $|i\rangle$, $|v\rangle$, $|ancillas\rangle$, t)
- 10 Encode(qc, $|k\rangle$, k)
- 11 inferieur(circuit, $|k\rangle$, $|v\rangle$, $|ancillas\rangle$, $|f_out\rangle$)
- 12 Encode(qc, $|k\rangle$, k)
- 13 Encodeur(qc, $|i\rangle$, $|v\rangle$, $|ancillas\rangle$, t)
- 14 amplification_grover(qc, $|i\rangle$, $|f_out\rangle$)
- 15 **fin**
- 16 opération de mesure sur le registre $|i\rangle$
- 17 qc.measure($|i\rangle$)
- 18 Exécution du circuit results \leftarrow execution(qc)
- 19 Fin de la partie s'exécutant sur la machine quantique
- 20 $min \leftarrow$ decode(results($|i\rangle$))
- 21 **si** $k \leq min$ **alors**
- 22 $debut \leftarrow min$
- 23 **sinon**
- 24 $fin \leftarrow min$
- 25 $e \leftarrow \frac{debut+fin}{2}$
- 26 **fin**
- 27 **retourner** e

Cet algorithme utilise le principe de la dichotomie et coupe l'espace de recherche en deux. Les valeurs possibles dans cet espace, ici un tableau, sont comprises entre 0 et D .

²Ce circuit n'est pas présenté dans ce mémoire. Il consiste seulement à écrire la chaîne binaire décrivant l'entier k avec des portes **X**.

La première étape consistera à trouver toutes les valeurs du tableau inférieur strict à $\frac{D}{2}$. S'il n'y en a aucune alors il faut chercher entre $\frac{D}{2} - 1$ et D sinon entre 0 et $\frac{D}{2} + 1$. Ces étapes de recherches seront répétées $O(\log_2(D))$. Supposons que D_{max} soit borné par le nombre de valeurs du tableau N , nous avons une complexité en $O(n\sqrt{\frac{N}{M}})$

Nous sommes à présent capable de trouver l'élément minimum (resp. maximum) dans une structure de données de type $T[i] = v$. Dans la section suivante, nous allons traiter le problème de minimisation sur un espace de recherche sous contraintes.

4.2 Minimiser une somme

L'algorithme du minimum nous permet d'obtenir l'élément minimum dans un ensemble, une liste ou un tableau. Présentons dans cette section comment minimiser une somme. Deux stratégies vont être étudiées. La première reprends une modélisation SAT du problème. La seconde stratégie utilise la relation entre l'affectation des variables entières et la valeur de la somme, comme les indexes et les valeurs d'un tableau.

Approche avec SAT

Commençons par présenter une première solution en utilisant la modélisation SAT. Comme nous avons déjà un circuit quantique pour SAT, il nous suffit juste de modéliser notre problème.

L'objectif est de trouver un encodage qui minimise au plus le nombre de variables et/ou le nombre de clauses afin de minimiser le nombre de qubits. Il existe plusieurs types d'encodages, l'encodage direct ou encore l'encodage logarithmique [21]. Nous avons choisi l'encodage le plus compact en raison du faible espace à disposition (32 qubits). Par exemple, si V est notre ensemble de variables et D le domaine, on aura besoin de $|V| \cdot |D|$ qubits avec l'encodage direct contre $|V| \log(max(D))$ qubits avec l'encodage logarithmique. Nous allons donc choisir l'encodage logarithmique qui réduit l'occupation en espace des variables.

Traitons l'exemple suivant avec l'encodage logarithmique. Soient, $V = \{x_1, x_2\}$, $D(x_1) = D(x_2) = \{1, 2, 3\}$ et la contrainte : $x_1 + x_2 \leq 4$. Comme $|V| = 2$ et $\log(max(D)) = 2$, nous avons un total de 2 variables pour encoder l'affectation dans D de chaque x_i . Nous aurons également besoin de variables intermédiaires pour calculer la valeur de somme de x_1 et x_2 . Comme cette somme peut atteindre 6 (3+3), il nous faut 3 variables s_1, s_2, s_3 et une variable jouant le rôle de retenue (carry) c_1 . Nous avons donc un total de 8 variables.

Nous allons avoir deux contraintes principales, celle du calcul de la somme, et celle de l'infériorité de la somme. Pour le calcul de la somme, il faut que $s_i = x_{1i} \vee x_{2i}$ pour retranscrire la somme binaire : $0 + 0 = 0$ et $1 + x = 1$, et pour le cas $1 + 1 = 0$, nous écrivons la clause $(\neg x_{1i} \vee \neg x_{2i} \vee \neg s_i)$ qui force s_i à être faux (0) si x_{1i} et x_{2i} sont à vrais (1+1). Enfin, dans le cas où x_{1i} et x_{2i} sont à vrai il faut transmettre la retenue ($c_1 = x_{11} \wedge x_{21}$) avec la clause : $(\neg x_{1i} \vee \neg x_{2i} \vee c_i)$. Enfin, pour la contrainte < 4 , il faut interdire s_3 qui représente la valeur 4.

- Variables : $x_{11}, x_{12}, x_{21}, x_{22}, s_1, s_2, s_3, c_1$

- Contrainte calcul de la somme :

$$- s_1 = x_{11} \vee x_{21} : (x_{11} \vee x_{21} \vee \neg s_1) \wedge (\neg x_{11} \vee \neg x_{21} \vee \neg s_1) \wedge (\neg x_{11} \vee x_{21} \vee s_1) \wedge (x_{11} \vee \neg x_{21} \vee s_1)$$

$$- c_1 = x_{11} \wedge x_{21} : (\neg x_{11} \vee \neg x_{21} \vee c_1)$$

$$- s_2 = x_{12} \vee x_{22} : (\neg x_{12} \vee \neg x_{22} \vee \neg c_1 \vee s_2) \wedge (\neg x_{12} \vee \neg x_{22} \vee c_1 \vee \neg s_2) \wedge (\neg x_{12} \vee x_{22} \vee \neg c_1 \vee \neg s_2) \wedge (x_{12} \vee \neg x_{22} \vee \neg c_1 \vee \neg s_2) \wedge (\neg x_{12} \vee x_{22} \vee c_1 \vee s_2) \wedge (x_{12} \vee \neg x_{22} \vee c_1 \vee s_2) \wedge (x_{12} \vee x_{22} \vee \neg c_1 \vee s_2) \wedge (x_{12} \vee x_{22} \vee c_1 \vee s_2)$$

$$- s_3 = x_{12} \wedge x_{22} : (\neg x_{12} \vee \neg x_{22} \vee s_3) \wedge (\neg x_{12} \vee \neg c_1 \vee s_3) \wedge (\neg x_{22} \vee \neg c_1 \vee s_3)$$

- Contrainte < 4 : $(\neg s_3)$

Avec l'encodage logarithmique, nous avons 8 variables et 15 clauses. Nous avons vu dans la partie SAT3.2 que nous avons besoin de $n+2m-1$ qubits pour résoudre une formule SAT. Pour traiter cette formule, il faut donc 37 qubits.

Malheureusement, nous ne pouvons pas modéliser un problème de sommes très grand. Introduisons à présent un circuit permettant d'additionner deux registres d'entiers non signés nécessaire à la réalisation de notre objectif : minimiser une somme.

La somme d'entiers

Dans [6] et [20] est décrit un algorithme de sommation basé sur une addition qubit à qubit. Nous allons utiliser ce circuit pour coder l'addition d'entiers.

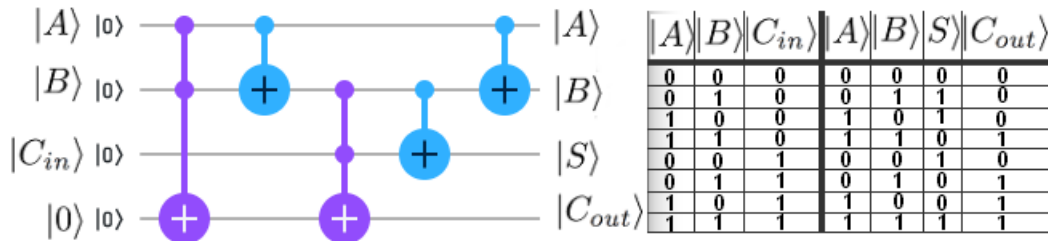


FIGURE 4.5 : Somme quantique

Dans ce circuit, nous additionnons le qubit $|A\rangle$ avec $|B\rangle$ en prenant compte du résultat précédant $|C_{in}\rangle$. Le résultat est contenu dans $|S\rangle$ et $|C_{out}\rangle$. Nous pouvons d'ailleurs considérer le dernier qubit de *carry* comme le dernier qubit du registre $|S\rangle$ étant le MSQ³. Si les deux qubits sommés sont dans l'état $|0\rangle$, alors aucune des **CX** n'est activées et le résultat est bien $|0\rangle$ ou dépendant du qubit $|C_{in}\rangle$. Si l'un des deux qubits est dans l'état $|1\rangle$, alors le qubit $|S\rangle$ sera dans l'état $|1\rangle$. En effet, si $|A\rangle$ seul est dans l'état $|1\rangle$, alors la deuxième **CX** va changer l'état de $|B\rangle$ en $|1\rangle$, qui lui aussi via une porte **CX** va changer l'état de $|S\rangle$ en $|1\rangle$. Dans l'autre cas, où seul $|B\rangle$ est dans l'état $|1\rangle$, la porte **CX** entre $|B\rangle$ et $|S\rangle$ va changer l'état de $|S\rangle$ de $|0\rangle$ à $|1\rangle$. Par contre, dans le cas où $|S\rangle$ serait déjà dans l'état $|1\rangle$ (cas où $|C_{in}\rangle$ est dans l'état $|1\rangle$) alors $|S\rangle$ passerait de l'état

³Qubit de poids le plus fort dans la chaîne du registre

$|1\rangle$ à l'état $|0\rangle$ et $|C_{out}\rangle$ passerait dans l'état $|1\rangle$ par l'application de la porte **CCX** entre $|B\rangle$ et $|C_{in}\rangle$. Le cas où $|A\rangle$ et $|B\rangle$ sont dans l'état $|1\rangle$ change dès la première **CCX** l'état du qubit $|C_{out}\rangle$ et comme la seconde porte **CX** change l'état de $|B\rangle$ à $|0\rangle$, il ne se passe plus rien dans le calcul. Pour se rassurer nous fournissons la table de vérité associé au circuit à droite de la figure 4.5.

L'algorithme général va appliquer cette opération sur tout le registre. Il est important de noter que si nous souhaitons faire des sommes successives, il faut prendre soin de réinitialiser les qubits de *carry* intermédiaires aux calculs de chaque somme sinon le résultat final serait faussé dès la deuxième somme.

Approche avec la somme d'entiers

Maintenant que nous avons un circuit codant l'addition, nous avons une façon de calculer une somme d'entiers. L'oracle que nous allons présenter évalue si le registre qui encode le résultat de la somme est le minimum en appelant le circuit *inférieur- k* vu précédemment. À chaque étape de recherche nous calculerons les affectations des variables telles que la somme est inférieure à k strictement. À la fin, nous obtiendrons le ou les affectations qui minimisent la somme.

Pour notre exemple : $x_1 + x_2 \leq 4$, nous avons besoin de 4 qubits pour les valeurs de x_1 et x_2 , 1 qubits de *carry* pour le calcul de somme. Enfin, il nous faut 2 qubits auxiliaires supplémentaires pour le calcul de l'infériorité et 1 qubit pour l'oracle, ce qui nous fait un total de 8 qubits.

Avec cet oracle nous pouvons modéliser des problèmes de sommes plus grand qu'avec la version SAT. Nous allons donc prendre cet approches pour la résolution des prochains problèmes.

4.3 Couverture de sommets de cardinalité minimale

Le problème de la couverture de sommets dans un graphe fait partie des 21 problèmes de Karp dans [13]. L'objectif va être de trouver un oracle permettant de couvrir toutes les arêtes d'un graphe $G = (V, E)$ tout en minimisant le nombre de sommets dans la couverture S . Nous pouvons reformuler le problème en une instance de SAT et utiliser l'algorithme décrit en section 3.2. En utilisant l'encodage SAT de la minimisation de somme nous pouvons obtenir une modélisation répondant au problème du *vertex-cover*.

Couverture de sommet : SAT

En reprenant la modélisation de la minimisation de somme, pour un graphe $G = (V, E)$, nous pouvons modéliser la contrainte : $\sum_{v \in V} x_v < k$, avec $x_v \in \{0, 1\}$ et k un entier. Enfin il manque la contrainte de couverture qui peut se modéliser ainsi : $\forall uv \in E, (x_u \vee x_v)$ ce qui contraint l'affectation d'au moins un sommet à 1 pour toutes les arêtes de G .

Comme nous avons vu précédemment que l'encodage de la contrainte $\sum_i x_i < k$ n'était pas optimal en le nombre d'utilisation des qubits, nous allons passer à une modélisation moins coûteuse en reprenant l'idée de la recherche du minimum dans le but de pouvoir exécuter notre circuit.

Oracle pour trouver une couverture minimale

La première idée est de représenter les arêtes comme un ensemble de clauses de taille 2 comme dans la sous-section précédente. Les sommets sont représentés par des variables tels que résoudre la formule 2-SAT revient à fournir une couverture (optimale ou pas) pour le graphe G .

Nous avons vu précédemment une façon d'obtenir le minimum d'une somme. À ce stade nous avons un ensemble de toutes les couvertures pour G fourni par la résolution de l'instance 2-SAT et nous voulons filtrer cet ensemble pour n'obtenir que les couvertures minimales. Nous allons donc appliquer l'algorithme de la recherche de minimum sur la somme des variables. Comme les variables représentent les sommets choisis, minimiser la somme des ces variables revient donc à minimiser les sommets choisis ce qui nous fournit finalement une couverture minimale pour G .

Algorithme 4 : Oracle : Couverture minimale

Input : circuit, $|nodes\rangle$, pour les sommets, $|aux\rangle$ pour les qubits auxiliaires, $|sum\rangle$ et $|carry\rangle$ pour la somme, $|k\rangle$ pour k , $|ancillas\rangle$, pour le registre d'ancillas, $G = (V, E)$ un graphe

Out : circuit : Circuit quantique

- 1 clauses $\leftarrow \square$
- 2 **pour chaque** $v_i v_j \in E$ **faire**
- 3 | clauses.add($v_i \vee v_j$) //Satisfaire la clause $v_i \vee v_j$, revient à couvrir $v_i v_j$
- 4 **fin**
- 5 Oracle_SAT(circuit, clauses, $|aux\rangle$, $|ancillas\rangle$)
- 6 somme($|nodes\rangle$, $|sum\rangle$, $|carry\rangle$) // $\sum_{i=0}^{|V|} v_i$
- 7 inferieur_k($|sum\rangle$, $|k\rangle$, $|f_{out}\rangle$)
- 8 **Unroll circuit**
- 9 **retourner** circuit

Cet algorithme code seulement la partie "Oracle" dans les étapes de l'algorithme de recherche de Grover. Il faut donc appeler l'oracle pour la couverture minimale comme sous procédure après l'initialisation et avant l'amplification.

Exécution

Dans l'exécution, ci-dessous, nous avons considéré le graphe C_4 , et pris pour valeur $k = 3$. Nous allons donc chercher toutes couvertures inférieures à $k = 3$. Nous avons besoin de 22 qubits et la taille du circuit est de 372 portes (unaire et binaire) sans transpilation.

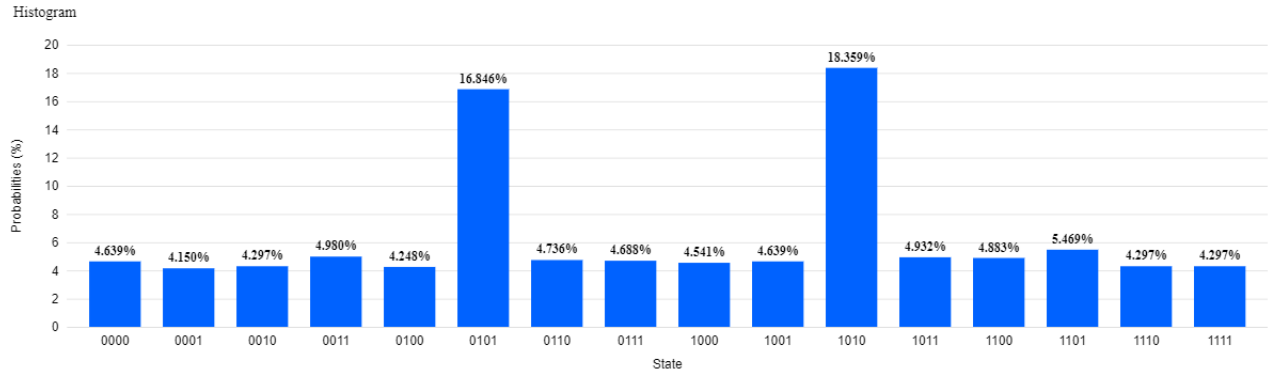
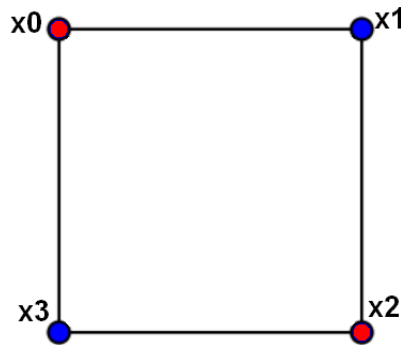


FIGURE 4.6 : Résultat sur simulateur

Après exécution sur le simulateur avec 2 itérations ($\sqrt{2^4/2} = 2$) et 1024 *shots*, celui-ci nous révèle deux solutions $|0101\rangle$ et $|1010\rangle$ pour nos qubits $x_3x_2x_1x_0$. Ces solutions sont illustrées sur la figure 4.8.

FIGURE 4.7 : Couverture de tailles 2 sur C_4 ($\{x_0, x_2\}$ et $\{x_1, x_3\}$).

Couverture de sommets pondérées

La version pondérée de la couverture de sommets est réalisable en minimisant non pas la somme des variables mais la somme de leurs poids. Afin de sommer les poids des sommets sélectionnés, nous allons devoir intriquer les poids avec la variable sommet tel que si la variable sommet est choisie et se trouve donc dans l'état $|1\rangle$, alors il faut ajouter le poids $|w_i\rangle$ à la somme finale. Dans le cas contraire nous ne faisons rien. Pour ce faire, nous pouvons également utiliser la somme contrôlée qui applique l'addition entre deux

registres seulement si le qubit de contrôle est dans l'état $|1\rangle$.

Algorithme 5 : Oracle : Couverture de sommets pondérée

Input : circuit, $|nodes\rangle$, pour les sommets, $|aux\rangle$ pour les qubits auxiliaires, $|k\rangle$ pour k , $|sum\rangle$ et $|carry\rangle$ pour la somme, $|w\rangle$ pour les poids, $|ancillas\rangle$, pour le registre d'ancillas, $G = (V, E)$ un graphe

Out : circuit : Circuit quantique

```

1 pour chaque  $uv \in E$  faire
2   | clauses.add( $u \vee v$ )
3 fin
4 pour chaque  $|n_i\rangle \in V$  faire
5   | encode(circuit,  $G[n_i]$ ,  $|w\rangle$ )
6   | Si  $n_i$  est choisi, à cet endroit du circuit on additionnera son
   | poids avec  $|sum\rangle$ 
7   | somme(circuit,  $|w\rangle$ ,  $|sum\rangle$ ,  $|carry\rangle$ ,  $|nodes_i\rangle$ )
8   | decode(circuit,  $G[n_0][n_1]$ ,  $|w\rangle$ )
9 fin
10 Oracle_SAT(circuit, clauses,  $|aux\rangle$ ,  $|ancillas\rangle$ )
11 inferieur_k( $|sum\rangle$ ,  $|k\rangle$ ,  $|f_{out}\rangle$ )
12 Unroll circuit
13 retourner circuit

```

Exécution

Poursuivons nos exécutions sur le même graphe C_4 avec cette fois-ci une pondération pour chaque sommet.

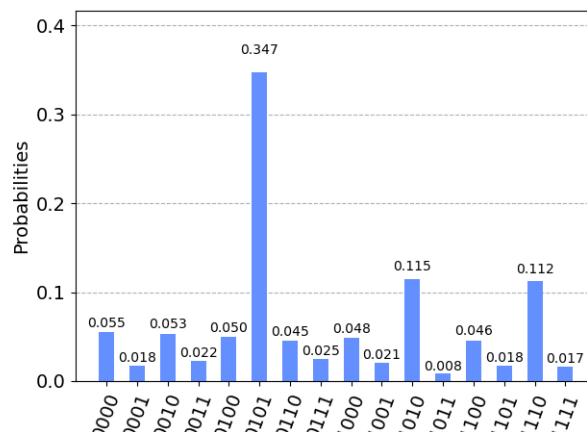


FIGURE 4.8 : Résultat sur simulateur avec les poids $[1, 3, 1, 3]$

Après exécution sur le simulateur nous remarquons qu'il ne reste plus qu'une solution : $|1010\rangle$, correspondant à la couverture de coût minimale, alors que sur la figure 4.6 nous avons obtenu deux résultats.

À présent, passons au dernier problème : le voyageur de commerce.

4.4 Voyageur de commerce

Le problème du voyageur de commerce est un cycle hamiltonien de poids minimal. Nous allons considérer les instances comme étant des graphes complets. La stratégie que nous allons utiliser va être de travailler sur une affectation d'une position dans le cycle pour chaque sommet. Comme ces positions doivent toutes être différentes nous allons avoir besoin de la contrainte *AllDifferent*.

Présentons dans un premier temps un circuit pour calculer la contraintes *AllDifferent* qui sera nécessaire pour le circuit final.

Contrainte : AllDifferent

Dans la figure ci dessous est présenté un exemple de circuit calculant si deux qubits, q_0 et q_1 sont différents. Le résultat est stocké sur q_3 , q_2 sert de registre auxiliaire pour le calcul.

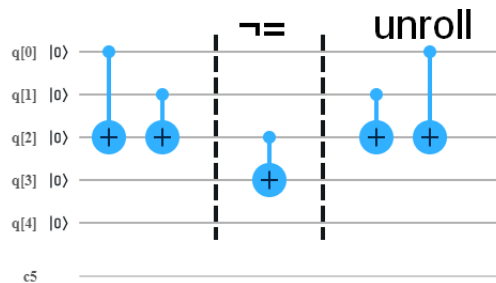


FIGURE 4.9 : Circuit calculant si q_0 est différent q_1

L'idée de ce circuit est de contrôler le qubit q_2 qui ne peut que être dans l'état $|1\rangle$ si l'un des qubit comparé au plus est dans l'état $|1\rangle$. En effet, si les deux qubits comparés sont dans l'état $|1\rangle$, l'application des deux **CX** mettraient q_2 dans l'état $|0\rangle$. De même si les deux qubits, q_0 et q_1 sont dans l'état $|0\rangle$, q_2 restera dans l'état $|0\rangle$. Finalement q_2 est dans l'état $|1\rangle$ seulement si q_0 est différent de q_1 . Comme q_2 est intriqué avec q_3 , le stockage du résultat, on va pouvoir réutiliser ce résultat pour d'autres calculs. Ce circuit se généralise avec des registres de plus grandes tailles.

Finalement on peut écrire un circuit qui calcule la contrainte *allDifferent* sur une

liste de registres de qubits.

Algorithme 6 : AllDifferent

Input : circuit, vars : liste de registres, $|aux\rangle$ pour les qubits auxiliaires, $|ancillas\rangle$, pour le registre d'ancillas, $|res\rangle$, pour stocker le résultat final

Out : circuit : Circuit quantique

- 1 **pour chaque** $|v_i\rangle \in vars$ **faire**
- 2 **pour chaque** $|v_j\rangle \in [i + 1, |vars|]$ **faire**
- 3 different(circuit, v_j , aux_k)
- 4 $k \leftarrow k + 1$
- 5 **fin**
- 6 **fin**
- 7 **N_CNOT**(circuit, $|aux\rangle$, $|ancillas\rangle$, $|res\rangle$)
- 8 **Unroll circuit**
- 9 **retourner** circuit

Nous pouvons utiliser cet algorithme en guise d'oracle pour trouver une affectation unique pour chaque variable. Dans le cas d'un graphe, si les variables représentent les positions des sommets, les résultats seront des cycles hamiltonien. À partir de ce circuit, nous allons pouvoir traiter le voyageur de commerce.

Oracle pour le voyageur de commerce

L'oracle pour trouver la solution au problème du voyageur de commerce utilise plusieurs sous-circuits vues précédemment. Nous avons déjà minimisé une somme de sommets pondérés en partie 5. Après avoir appliqué la contrainte *AllDifferent*, nous devons seulement sommer les poids des arêtes parcourues. Une arête (uv) est parcourue seulement si $pos_u - pos_v = 1$. En utilisant un circuit calculant la différence entières (\mathbb{N}) sur les positions nous allons pouvoir sommer les arêtes parcourues une seule fois. Enfin, nous testerons avec le circuit "equal" si la soustraction entière est égale à 1 et le résultat de ce test sera utilisé comme "contrôle" de la somme. Une fois la somme calculée, il faut vérifier si elle est minimale en appliquant le circuit *inférieur_k* en appelant l'oracle un

nombre suffisant (voir la section recherche du minimum 4.1).

Algorithme 7 : Oracle : Voyageur de commerce

Input : circuit, $|nodes\rangle$, pour les sommets, $|w\rangle$ pour les poids, $|aux\rangle$ pour les qubits auxiliaires, $|ctrl\rangle$ le qubit de contrôle, $|res\rangle$ pour la soustraction, $|sum\rangle$ et $|carry\rangle$ pour la somme, $|k\rangle$ le poids max du cycle, et $|ancillas\rangle$, pour le registre d'ancillas, $G = (V, E)$ un graphe pondéré, k le poids max du cycle autorisé

Out : circuit : Circuit quantique

- 1 encode(circuit, k , $|k\rangle$)
- 2 $|aux_{m-1}\rangle$ est l'avant dernier qubit du registre $|aux\rangle$
- 3 allDifferent(circuit, $|nodes\rangle$, $|aux\rangle$, $|anc\rangle$, $|aux_{m-1}\rangle$)
- 4 **pour chaque** $|node_i\rangle \in |nodes\rangle$ **faire**
- 5 **pour chaque** $|node_{j \neq i}\rangle \in |nodes\rangle$ **faire**
- 6 sub($|node_i\rangle$, $|node_j\rangle$, $|res\rangle$)
- 7 equal($|res\rangle$, 1, $|ctrl\rangle$)
- 8 encode(circuit, $G[i][j]$, $|w\rangle$)
- 9 ctrl_sum(circuit, $|w\rangle$, $|sum\rangle$, $|carry\rangle$, $|ctrl\rangle$)
- 10 Réinitialisation pour le prochain calcul
- 11 unroll_encode(circuit, $G[i][j]$, $|w\rangle$)
- 12 unroll_equal($|res\rangle$, 1, $|ctrl\rangle$)
- 13 unroll_sub($|node_i\rangle$, $|node_j\rangle$, $|ctrl\rangle$)
- 14 **fin**
- 15 **fin**
- 16 $|aux_{m-1}\rangle$ est l'avant dernier qubit du registre $|aux\rangle$
- 17 inférieur_k(circuit, $|sum\rangle$, $|k\rangle$, $|aux\rangle$, $|anc\rangle$, $|aux_m\rangle$)
- 18 on contrôle les contraintes, allDiff(x_0, \dots, x_n) et $\sum_{i,j} w_{ij} < k$
- 19 circuit.CCX($[|aux_{m-1}\rangle, |aux_m\rangle]$, f_{out})
- 20 Unroll circuit
- 21 **retourner** circuit

Le circuit complet serait donc cet oracle entouré des deux autres étapes de Grover, l'initialisation et l'amplification en plus de la partie séquentiel de la recherche du minimum dichotomique.

Pour une instance comme K_4 , notre circuit nécessite 35 qubits⁴.

Nous n'avons pas réussi à obtenir un résultat sur simulateur avec les méthodes d'exécutions classiques pour le voyageur de commerce. Le temps d'exécution est trop long et dépasse le *time_out* du simulateur fixé à 10 000 s 4.10.

⁴On peut encore descendre le nombre de qubits en utilisant des **N_CNOT** optimisés de Qiskit n'utilisant pas d'ancilla.

Status	Time taken
Job timed out after 10000 seconds. [5201]	2h 47m 8.1s

FIGURE 4.10 : Temps d'exécutions supérieur à 10000 s.

Nous avons également essayé d'exécuter le circuit sur la machine *ibmq_rochester* à 53 qubits mais le circuit est trop grand et n'a donc pas pu être exécuté.

Complexité

Nous avons vu que la complexité de la recherche du minimum dichotomique était de $O(n\sqrt{N}) = O(n2^{\frac{n}{2}})$ 3. La complexité du voyageur de commerce quantique est donc de $O(n2^{\frac{n}{2}})$, hors en 1962 Held et Karp on fournit un algorithme en programmation dynamique en $O(n^2.2^n)$. Nous avons donc une accélération quadratique d'un algorithme de voyageur de commerce complet grâce à Grover.

Conclusion

L'arrivée des machines quantiques physiques permettent de concrétiser la théorie. Malgré que les machines soient encore trop bruitées pour exploiter les résultats des calculs, des simulateurs permettent d'évaluer le comportement des futures implémentations d'algorithmes. Par exemple, le circuit quantique dérivé de l'algorithme de Grover permet de résoudre le problème SAT avec une complexité de $O(\sqrt{N})$.

Il sera dès lors possible de résoudre un très grand nombre de problèmes quand le coût en espace sera négligeable. Selon les prévisions d'IBM nous pouvons envisager, une fois le bruit stabilisé, l'utilisation des circuits quantiques de plus en plus grands.

Les machines Tenerife, Tokyo, Johannesburg et Raleigh sont des machines d'IBM avec un volume quantique de 32 et la machine de Paris que nous avons régulièrement utilisé a également un volume quantique de 32. Enfin, Honeywell a annoncé le 3 mars 2020 avoir une machine avec un volume quantique de 64 qu'ils ont livré Vendredi 19 Juin.

Dans ce rapport, nous avons commencé en décrivant les aspects physiques de l'informatique quantique pour petit à petit quitter ce monde et entrer dans celui de l'informatique à partir de la section 3.2 avec le problème SAT. En plus d'obtenir des algorithmes avec une accélération par rapport à leur version séquentielle, nous n'avons pas réussi à remarquer de différence en terme de complexité entre trouver une ou toutes les solutions.

En plus d'un *solver* SAT quantique permettant de résoudre les modélisations SAT existantes, nous avons également présenté les circuits de bases pour faire de l'arithmétique (addition, soustraction) ainsi que des opérations logiques ($<$, $>$, \neq , $=$), le tout permettant de faire des circuits traitant des problèmes d'optimisation combinatoire.

Ce stage m'a permis de découvrir le monde de la recherche qui m'était encore inconnu. J'ai également pu découvrir l'univers de l'informatique quantique que je trouve très intéressant. On y retrouve toutes les problématiques de début de l'ère de l'informatique (précision des machines, écriture de programme sur de très faible capacité de mémoire).

Bibliographie

- [1] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6) :525–532, November 1973.
- [2] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5) :1411–1473, 1997.
- [3] G. Brassard and P. Hoyer. An exact quantum polynomial-time algorithm for simon’s problem. *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*.
- [4] Bogusław Broda. Quantum search of a real unstructured database. *The European Physical Journal Plus*, 131(2), Feb 2016.
- [5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC ’71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.
- [6] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit, 2004.
- [7] P. A. M. Dirac. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 35(3) :416–418, 1939.
- [8] Paul Dirac. *The Principles of Quantum Mechanics*, volume 1. Clarendon Press, 1930.
- [9] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum, 1996.
- [10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
- [11] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.
- [12] Lov K. Grover. Quantum Computers Can Search Rapidly by Using Almost Any Transformation. 80(19) :4329–4332, May 1998.

- [13] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.
- [14] Giacomo Nannicini. An introduction to quantum computing, without the physics, 2017.
- [15] Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information*, volume 1. Cambridge University Press, 2010.
- [16] David Oliveira and Rubens Ramos. Quantum bit string comparator : Circuits and applications. *Quantum Computers and Computing*, 7, 01 2007.
- [17] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), Jul 2014.
- [18] E. Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Phys. Rev.*, 28 :1049–1070, Dec 1926.
- [19] Qiskit Team. Qiskit, Jun 2020.
- [20] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1) :147–153, Jul 1996.
- [21] Toby Walsh. Sat v csp. In Rina Dechter, editor, *Principles and Practice of Constraint Programming – CP 2000*, pages 441–456, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.