# Projet IQRO 2026

Alexandre PHONCHAREUN, Julien LIN, Paul PASSERON, Rayan MERKHI

January 2026

## 1 Description du Problème

Soit $A \in \{0,1\}^{m \times n}$ une matrice binaire et $b \in \{0,1\}^m$ un vecteur cible. Nous cherchons un vecteur binaire $x \in \{0,1\}^n$ qui maximise le nombre d'égalités satisfaites dans le système :

$$Ax \equiv b \pmod 2 \tag{1}$$

Ce problème est connu sous le nom de **MAX-XOR-SAT** ou encore le problème de décodage par maximum de vraisemblance pour des codes linéaires.

Contrairement à la résolution exacte d'un système linéaire (qui peut se faire en temps polynomial via l'élimination de Gauss-Jordan), chercher à satisfaire le *maximum* d'équations dans un système surdéterminé ou incohérent est un problème **NP-Difficile**.

## 2 Description du problème et question préliminaire

**3. Soit $\oplus$ l'opérateur XOR. Trouver un polynôme de degré 2, utilisant les variables $x_1$ et $x_2$ égal à $x_1 \oplus x_2$**

On remarque que $\forall (x_1, x_2) \in \{0,1\}^2$,

$$(1 - x_1)x_2 + (1 - x_2)x_1 = x_1 + x_2 - 2x_1x_2 = x_1 \oplus x_2$$

**4. Généralisez à $\bigoplus_{i=1}^{n} x_i$. Il faudra utiliser un polynôme de degré $n$.**

Soit bit $\in B_n$ où $B_n$ est l'ensemble des *bitstrings* de taille $n$. On note $\text{bit}_i \in \{0,1\}$ le $i$-ème bit de bit. Notons $P_{\text{bit}}$ le monôme associé à bit, t el que:

$$P_{\text{bit}}(X_1, \ldots, X_n) = \prod_{\substack{k=1 \\ \text{bit}_k=1}}^{n} X_k \prod_{\substack{k=1 \\ \text{bit}_k=0}}^{n} (1 - X_k)$$

Ainsi, $P_{\text{bit}}(X_1, \ldots, X_n) = 1$ ssi. $\forall 1 \leq k \leq n, X_k = \text{bit}_k$ et 0 sinon.

On obtient donc un "filtre" pour chaque *bitstring*.

Le principe est maintenant d'énumérer chaque *bitstring* dont le XOR successifs de chaque bit vaut 1 et de les additionner.

On note que $\bigoplus_{i=1}^{n} \text{bit}_i = 1$ ssi. $\sum_{i=1}^{n} \text{bit}_i \equiv 1 \pmod 2$. On nomme $\text{Impair}_n$ l'ensemble des tels *bitstrings*.

Finalement, on a

$$\text{XOR}(X_1, \ldots, X_n) = \sum_{\text{bit } \in \text{ Impair}_n} P_{\text{bit}}(X_1, \ldots, X_n) \tag{2}$$

$$\text{XOR}(X_1, \ldots, X_n) = \sum_{\text{bit } \in \text{ Impair}_n} \left( \prod_{\substack{k=1 \\ \text{bit}_k=1}}^{n} X_k \prod_{\substack{k=1 \\ \text{bit}_k=0}}^{n} (1 - X_k) \right) \tag{3}$$

## 3 Résolution avec QAOA.

**5. Reformuler le problème de Max XOR SAT sous forme HOBO. Aidez vous de la question 4. Attention au cas où $b_i$= 1.**

Un problème de **QAOA** en **HOBO** à la forme suivante:

$$\min_{(x_1,\ldots,x_n)\in\{0,1\}^n} \sum_{i_1,\ldots,i_n=0}^{n} c_{i_1,\ldots,i_n} \left(x_{i_1} \times \cdots \times x_{i_n}\right) \tag{4}$$

Dans notre cas, on veut maximiser le nombre d'équations vérifiées, c'est à dire minimiser le nombre d'équations fausses.

On note $b_i$ la $i$-ème coordonnée de $b$ et $a_{i,j}$ la valeur de coordonnée $(i, j)$ de $A$.

Ainsi, la fonction à minimiser peut s'écrire

$$f_n(x) = \sum_{j=1}^{m} |\text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) - b_j| \tag{5}$$

De plus, $\forall i \in [|1, n|],\ j \in [|1, m|]$, on a:

- Lorsque $b_j = 0$.

$$|\text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) - b_j| = \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) \tag{6}$$

- Lorsque $b_j = 1$.

$$|\text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) - b_j| = 1 - \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) \tag{7}$$

On a donc

$$f_n(x) = \sum_{\substack{j=1 \\ b_j=0}}^{m} \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) + \sum_{\substack{j=1 \\ b_j=1}}^{m} 1 - \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) \tag{8}$$

On peut donc finalement formuler le problème **Max XOR SAT** sous la forme HOBO suivante:

$$\min_{(x_1,\ldots,x_n)\in\{0,1\}^n} \sum_{\substack{j=1 \\ b_j=0}}^{m} \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) + \sum_{\substack{j=1 \\ b_j=1}}^{m} 1 - \text{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) \tag{9}$$

où on a bien un polynôme de degré $n$.

**6. Effectuer un changement de variable pour remplacer les variables binaires par des variables $\{-1, 1\}$**

On note le changement de variable $z = x \mapsto 1 - 2x$ qui à 0 associe 1 et à 1 associe $-1$.

On note $\forall i \in [|1, n|]$,

$$z_i = 1 - 2x_i \tag{10}$$

En appliquant ce changement de variable à l'expression obtenu en question 4 mènerai à un polynôme difficilement exploitable.

Pour cette raison revenons à l'expression du XOR sur 2 bits trouvé en question 3 :

$$x_1 \oplus x_2 = (1 - x_1)x_2 + (1 - x_2)x_1 = x_1 + x_2 - 2x_1x_2$$

En appliquant le changement de variable et en simplifiant l'expression, on trouve :

$$\text{XOR}(z_1, z_2) = \frac{1}{2}(1 - z_1z_2) \tag{11}$$

On peut ainsi en déduire par récurrence :

$$\mathrm{XOR}(z_1, \ldots, z_n) = \frac{1}{2}\left(1 - \prod_{i=1}^{n} z_i\right) \tag{12}$$

**7. Écrire l'Hamiltonien $H_C$ associé, montrer sur un exemple avec 3 variables et 2 contraintes de votre choix qu'une solution est bien associée à un vecteur propre de valeur propre $\lambda$ égale à l'opposé du nombre de contraintes non satisfaites.**

On note l'opérateur de Pauli $\widehat{Z}_k$ agissant sur le qubit $k$.

On a maintenant

$$f_n(x) = \sum_{\substack{j=1 \\ b_j=0}}^{m} \mathrm{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) + \sum_{\substack{j=1 \\ b_j=1}}^{m} 1 - \mathrm{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) \tag{13}$$

Par simplicité, on va définir

$$\begin{aligned} \mathrm{XOR}_j(x_1, \ldots, x_n) &= \mathrm{XOR}(a_{1,j}x_1, \ldots, a_{n,j}x_n) && (14)\\ &= \frac{1}{2}\left(1 - \prod_{i=1}^{n}(1 - 2a_{i,j}x_i)\right) && (15)\\ &= \frac{1}{2}\left(1 - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j}(1 - 2x_i)\right) && (16) \end{aligned}$$

Avec

$$n_j = |\{k \in [|1, n|], a_{k,j} = 1\}| \tag{17}$$

Ce qui nous donne donc

$$f_n(x) = \sum_{\substack{j=1 \\ b_j=0}}^{m} \mathrm{XOR}_j(x_1, \ldots, x_n) + \sum_{\substack{j=1 \\ b_j=1}}^{m} 1 - \mathrm{XOR}_j(x_1, \ldots, x_n) \tag{18}$$

En effectuant le changement de variable comme dans la question précédente, on obtient alors:

$$\mathrm{XOR}_j(z_1, \ldots, z_n) = \frac{1}{2}\left(1 - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} z_i\right) \tag{19}$$

On peut ainsi écrire $f_n$ en fonction des $z$:

$$f_n(z) = \sum_{\substack{j=1 \\ b_j=0}}^{m} \mathrm{XOR}_j(z_1, \ldots, z_n) + \sum_{\substack{j=1 \\ b_j=1}}^{m} 1 - \mathrm{XOR}_j(z_1, \ldots, z_n) \tag{20}$$

On a ainsi

$$H_C = \sum_{\substack{j=1 \\ b_j=0}}^{m} \widehat{\mathrm{XOR}}_j + \sum_{\substack{j=1 \\ b_j=1}}^{m} \left(\mathbb{I} - \widehat{\mathrm{XOR}}_j\right) \tag{21}$$

Avec $\widehat{\mathrm{XOR}}_j$ l'opérateur obtenu en remplaçant les $z_i$ par $\widehat{Z}_i$ dans la formule de $\mathrm{XOR}_j$ pour tout $j$:

$$\widehat{\mathrm{XOR}}_j = \frac{1}{2}\left(\mathbb{I} - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i\right) \tag{22}$$

On obtient alors l'Hamiltonien:

$$H_C = \sum_{\substack{j=1 \\ b_j=0}}^{m} \widehat{\mathrm{XOR}}_j + \sum_{\substack{j=1 \\ b_j=1}}^{m} \left( \mathbb{I} - \widehat{\mathrm{XOR}}_j \right) \tag{23}$$

$$= \sum_{\substack{j=1 \\ b_j=0}}^{m} \frac{1}{2} \left( \mathbb{I} - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) + \sum_{\substack{j=1 \\ b_j=1}}^{m} \left( \mathbb{I} - \frac{1}{2} \left( \mathbb{I} - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \right) \tag{24}$$

$$= \sum_{\substack{j=1 \\ b_j=0}}^{m} \frac{1}{2} \left( \mathbb{I} - \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) + \sum_{\substack{j=1 \\ b_j=1}}^{m} \frac{1}{2} \left( \mathbb{I} + \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{25}$$

$$= \sum_{j=1}^{m} \frac{1}{2} \left( \mathbb{I} - (-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{26}$$

Finalement, on a:

$$H_C = \frac{m}{2}\mathbb{I} - \frac{1}{2} \sum_{j=1}^{m} (-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \tag{27}$$

**On peut vérifier l'expression de l'hamiltonien avec un exemple:**

Prenons $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$, $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ et $b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, on obtient le système suivant :

$$\begin{cases} x_1 + x_2 = 0 \pmod 2 \\ x_2 + x_3 = 1 \pmod 2 \end{cases}$$

L'hamiltonien associé à ce système est donc :

$$H_C = \frac{1}{2}(\mathbb{I} - \widehat{Z}_1\widehat{Z}_2) + \frac{1}{2}(\mathbb{I} + \widehat{Z}_2\widehat{Z}_3)$$

Vérifions, avec plusieurs solutions, que le vecteur propre associé à la solution est bien de valeur propre $\lambda$ égale au nombre de contraintes **non-satisfaite**.

- Soit une première solution $x = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$

On a :

$$\begin{cases} x_1 + x_2 = 1 + 1 = 0 \pmod 2 \\ x_2 + x_3 = 1 + 0 = 1 \pmod 2 \end{cases}$$

Donc $x = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ est une solution "parfaite" qui satisfait les 2 contraintes. On a donc dans ce cas 0 contrainte non-satisfaite et donc la valeur propre du vecteur propre $|110\rangle$ devrait être 0.

Vérifions cela:

On rappelle que $Z|0\rangle = |0\rangle$ et $Z|1\rangle = -|1\rangle$

Ici $\widehat{Z}_k$ est l'opérateur $Z$ agissant sur le qubit $k$.

$$\begin{aligned} H_C|110\rangle &= \frac{1}{2}(\mathbb{I} - \widehat{Z}_1\widehat{Z}_2)|110\rangle + \frac{1}{2}(\mathbb{I} + \widehat{Z}_2\widehat{Z}_3)|110\rangle && (28) \\ &= \frac{1}{2}(|110\rangle - \widehat{Z}_1\widehat{Z}_2|110\rangle) + \frac{1}{2}(|110\rangle + \widehat{Z}_2\widehat{Z}_3|110\rangle) && (29) \\ &= \frac{1}{2}(|110\rangle - |110\rangle) + \frac{1}{2}(|110\rangle - |110\rangle) && (30) \\ &= 0|110\rangle && (31) \end{aligned}$$

$|110\rangle$ a bien comme valeur propre 0, qui est bien le nombre de contrainte non-satisfaite par la solution.

- Soit une deuxième solution $x = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

Cette solution vérifie 1 contrainte (la deuxième), donc on a 1 contrainte non satisfaite. La valeur propre associée à $|101\rangle$ devrait donc être 1.

$$H_C \left|101\right\rangle = \frac{1}{2}(\mathbb{I} - \widehat{Z}_1\widehat{Z}_2)\left|101\right\rangle + \frac{1}{2}(\mathbb{I} + \widehat{Z}_2\widehat{Z}_3)\left|101\right\rangle \tag{32}$$

$$= \frac{1}{2}(\left|101\right\rangle - \widehat{Z}_1\widehat{Z}_2\left|101\right\rangle) + \frac{1}{2}(\left|101\right\rangle + \widehat{Z}_2\widehat{Z}_3\left|101\right\rangle) \tag{33}$$

$$= \frac{1}{2}(\left|101\right\rangle + \left|101\right\rangle) + \frac{1}{2}(\left|101\right\rangle - \left|101\right\rangle) \tag{34}$$

$$= 1\left|101\right\rangle \tag{35}$$

On a bien 1 comme valeur propre pour $|101\rangle$.

- Essayons une dernière solution $x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

Avec cette solution il est assez évident que les 2 contraintes sont non-satisfaites. On devrait donc trouver une valeur propre 2 pour le vecteur $|100\rangle$

$$H_C \left|100\right\rangle = \frac{1}{2}(\left|100\right\rangle - \widehat{Z}_1\widehat{Z}_2\left|100\right\rangle) + \frac{1}{2}(\left|100\right\rangle + \widehat{Z}_2\widehat{Z}_3\left|100\right\rangle) \tag{36}$$

$$= \frac{1}{2}(\left|100\right\rangle + \left|100\right\rangle) + \frac{1}{2}(\left|100\right\rangle + \left|100\right\rangle) \tag{37}$$

$$= 2\left|100\right\rangle \tag{38}$$

On a bien 2 comme valeur propre pour $|100\rangle$.

**8. Décrire le circuit quantique de la matrice unitaire** $\exp i\gamma H_C$

Soit $\gamma$ une phase. On définit

$$U_C(\gamma) = \exp i\gamma H_C \tag{39}$$

$$= \exp i\gamma \left( \frac{m}{2}\mathbb{I} - \frac{1}{2}\sum_{j=1}^{m}(-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{40}$$

$$= \exp\left(\frac{i\gamma m}{2}\mathbb{I}\right) \exp i\gamma \left( -\frac{1}{2}\sum_{j=1}^{m}(-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{41}$$

$$= e^{\left(\frac{i\gamma m}{2}\right)} \exp i\gamma \left( -\frac{1}{2}\sum_{j=1}^{m}(-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{42}$$

On note que les $(\widehat{Z}_i)_i$ commutent toutes deux à deux. on peut écrire:

$$U_C(\gamma) = e^{\left(\frac{i\gamma m}{2}\right)} \prod_{j=1}^{m} \exp\left( -i\frac{\gamma}{2}(-1)^{b_j} \prod_{\substack{i=1 \\ a_{i,j}=1}}^{n_j} \widehat{Z}_i \right) \tag{43}$$

On note que la phase globale de $e^{\left(\frac{i\gamma m}{2}\right)}$ n'est pas utile au circuit.

Finalement, l'opérateur de coût $U_C(\gamma)$ correspondant au circuit se décompose en un produit de portes unitaires correspondant à chaque contraint XOR. Chaque contrainte est associée à une exponentielle d'un produit de matrices de Pauli $Z$, implémentable par une suite de portes CNOT (pour la parité) suivie d'une rotation $R_Z$ (pour la phase). Les termes du Hamiltonien commutant entre eux, l'ordre est indifférent.

# 4 Résolution avec l'algorithme de Grover

## 4.1 Résolution du problème de décision associé

### 10. Écrire le problème de décision associé au problème Max XOR SAT

Soit une instance définie par une matrice $A \in M_{m,n}\{0,1\}$, un vecteur $b \in \{0,1\}^m$ et un entier k (seuil de satisfaction).

Existe-t-il un vecteur $x \in \{0,1\}$ tel que le nombre de contraintes satisfaites par $x$ dans le système découlant de $Ax = b \pmod 2$ est au moins égal à k ?

L'oracle de Grover devra donc marquer les états $|x\rangle$ qui respectent cette condition.

En faisant monter petit à petit k, on pourra donc maximiser le nombre d'équations (contraintes) satisfaites pour une instance du problème Max XOR SAT.

**11. Proposer schématiquement la construction de l'oracle du problème de décision, en précisant le rôle de chaque partie de l'oracle.**

L'oracle $O_f$ doit effectuer l'opération $O_f |x\rangle = -|x\rangle$ si x satisfait au moins k contraintes et $O_f |x\rangle = |x\rangle$ sinon.

Pour encoder ces informations, on aura besoin de 2 registres. Le premier registre sera le registre d'entrée, encodant les valeurs du vecteur candidat $x$. Le deuxième registre sera un registre d'ancillas $|c_i\rangle$ de $m$ qubits : on flip le qubit $|c_i\rangle$ à $|1\rangle$ si la solution $x$ satisfait la contrainte $i$ sinon on laisse à $|0\rangle$. Cette étape du circuit dépend de l'instance du problème mais globalement, cette vérification peut être accompli via des portes $CNOT$ contrôlées par les $x_j$ vers $c_i$ suivies d'une porte $X$ en fonction de si $b_i$ est à 0 ou 1.

Finalement, on applique une porte qui prend en entrée le registre des $|c_i\rangle$ et vérifie si le nombre de qubit à 1 est supérieur ou égale à $k$. Si oui, elle applique un changement de phase sur $|x\rangle$ via un qubit cible (initialisé à $|-\rangle$) sur lequel on applique une porte $X$ et grâce au principe de phase kickback.

On rappelle que $X|-\rangle = -|-\rangle$

Remarque : il faudrait aussi remettre les qubits ancillas à $|0\rangle$ en appliquant les mêmes opérations à l'envers.

**12. Proposer un circuit pour l'instance suivante et la valeur k = 2. Vous pouvez supposer que vous disposer d'une porte In qui indique si au moins k qbits parmi ses entrées sont égales à 1 (le nombre d'entrées, et la description des sorties de cette porte peuvent être indiquées librement dans votre réponse).**

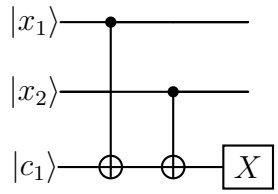
Avec $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$ , $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ et $b = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ , on obtient le système suivant :

$$\begin{cases} x_1 + x_2 = 0 \pmod 2 \\ x_2 = 1 \pmod 2 \\ x_1 = 0 \pmod 2 \end{cases}$$

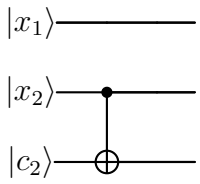
L'objectif est donc de marquer les solutions qui vérifient au moins $k = 2$ de ces équations (contraintes).

Ici, l'oracle va nécessiter 3 registres. Le premier registre aura 2 qubits pour encoder les 2 variables de $x$ : $x_1$ et $x_2$. Le deuxième registre aura 3 qubits : $c_1$, $c_2$ et $c_3$ correspondant à chaque contrainte et qu'on initialise à $|0\rangle$. Le qubit $c_i$ sera mis à $|1\rangle$ si la contrainte $i$ est satisfaite. Le dernier registre aura 1 qubit initialisé à $|-\rangle$ et permettra le changement de phase si la solution proposé $(x_1, x_2)$ permet d'obtenir au moins $k = 2$ satisfaites.

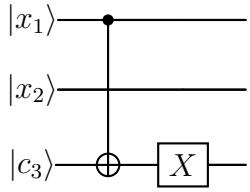
La première contrainte $x_1 + x_2 = 0 \pmod 2$ traduit que la somme doit être paire. On remarque donc que cette contrainte n'est vérifié que si $|x_1\rangle$ et $|x_2\rangle$ ont la même valeur. Pour traduire cela, on peut appliquer 2 portes $CNOT$ contrôlées par $|x_1\rangle$ puis $|x_2\rangle$ qui permet de vérifier si les 2 qubits sont différents. Etant donné que l'on veut que les 2 qubits soit identiques, on inverse le résultat avec une porte $X$ sur $|c_1\rangle$.



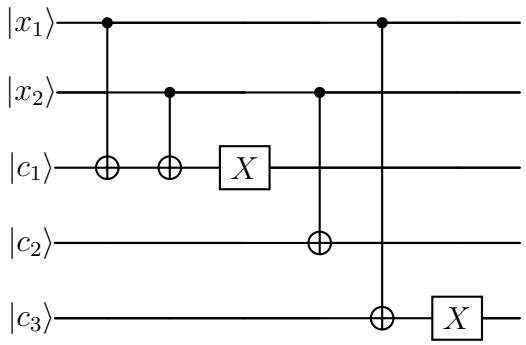

Pour la deuxième contrainte, c'est plus direct. On a seulement besoin de vérifier si $|x_2\rangle$ est à 1. Une simple $CNOT$ contrôlée par $|x_2\rangle$ le permet.



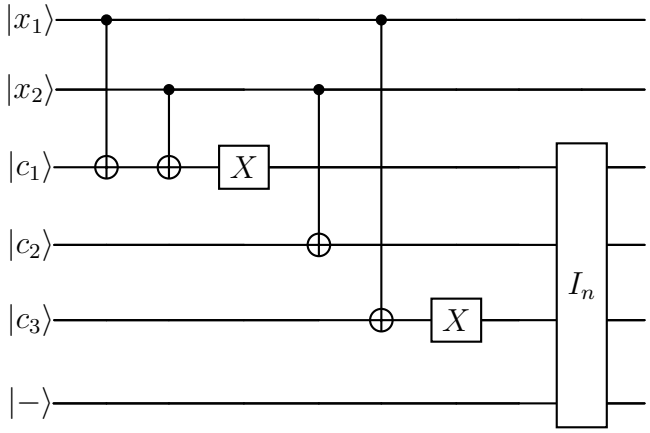

La troisième contrainte est presque identique à la deuxième mais ici on veut vérifier si $|x_1\rangle$ est à 0. Il faut donc ajouter une porte $X$ pour inverser le résultat.




On a donc comme première partie de circuit :



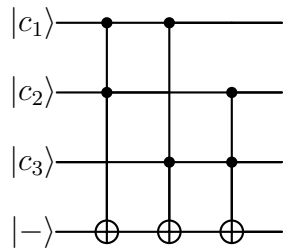

A partir de là, on a plus qu'à ajouter la porte $In$ qui prend en entrée les $|c_i\rangle$ et applique une $X$ sur le qubit cible si au moins 2 des $|c_i\rangle$ sont à 1 (i.e si au moins 2 contraintes sont satisfaites) engendrant le changement de phase sur le registre $|x\rangle$ (phase kickback).




Ce circuit nous permet donc de vérifier si une solution $(x_1, x_2)$ satisfait au moins $k = 2$ équations pour cette instance du problème.

Remarque : il faudrait remettre les qubits ancillaires $|c_i\rangle$ à $|0\rangle$ en ré-appliquant les opérations dans le sens inverse.

On peut aussi détailler le circuit de la porte $In$ pour cette instance qui est relativement simple :




**13. Pour ne pas exploser le nombre de qubits avec les qubits de travail dans Qiskit en construisant la porte $I_n$, on va tricher un peu. Soit une liste $L$ de $p$ nombres entre $0$ et $2^n - 1$, expliquer comment construire un circuit de profondeur $O(|L|)$ qui, connaissant un qubit de base $|x\rangle$, renvoie $-|x\rangle$ si $x \in L$ et renvoie $|x\rangle$ sinon.**

Tout d'abord, il est nécessaire de créer une porte contrôlé pour chaque nombre y appartenant à L. Soit $y \in L$, on défini $C_y$ par :

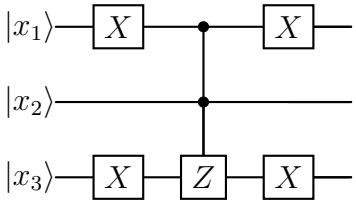
$$C_y |x\rangle = \begin{cases} -|x\rangle & \text{si } x = y \\ |x\rangle & \text{sinon} \end{cases}$$

Pour créer cette porte contrôlé, il faut d'abord appliquer une porte X pour chaque qbit de $y$ à $|0\rangle$ ainsi si le qubit $j$ de $x$ est égal au qubit $j$ de $y$, alors nous obtenons forcément en sortie $|1\rangle$ pour le qubit $j$ de $x$.

Ainsi si $|x\rangle = |y\rangle$ alors on obtient $|111...1\rangle$, si $|x\rangle \neq |y\rangle$ alors on aurait au moins un 0 dans l'état de sortie.

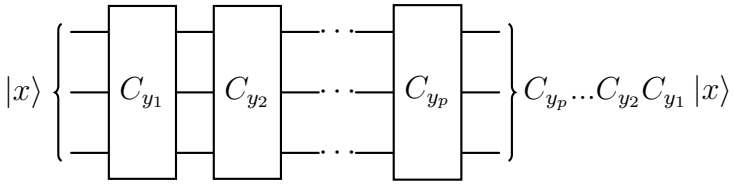
Ensuite, nous avons 2 cas : si nous avons un état $|111...1\rangle$ (correspondant à $|x\rangle$=$|y\rangle$) nous devons appliquer une inversion de phase, sinon si nous avons un état avec au moins 1 qubit à 0 (donc $|x\rangle \neq |y\rangle$) et nous ne faisons rien. Pour cela, nous appliquons une porte multi contrôlé Z sur tout les qubits, celle-ci fait exactement ce qu'on désire. Enfin il suffit d'appliquer de la même manière les portes X du début pour retrouver $|x\rangle$.

Voici un exemple à 3 qubits pour $y = 010$ :




Maintenant que nous avons construit un circuit qui "marque" une solution $y_i$ de L, il suffit d'appliquer successivement les $p$ portes $C_{y_i}$ correspondant à toutes les solutions de L pour construire un circuit qui inversera la phase de tout état $|x\rangle$ telle que $x \in L$. En effet, chaque porte $C_{y_i}$ n'agira que si et seulement si $x = y_i$.

Ainsi le circuit général ressemble à cela :




Chaque porte $C_y$ est de profondeur 3 et nous l'appliquons p fois. Donc nous obtenons bien un circuit de profondeur $O(p) = O(|L|)$ qui fait en sorte de nous donner -$|x\rangle$ si il existe $y \in L$ tel que $|x\rangle = |y\rangle$

**15. Expliquer pourquoi concevoir ce circuit est en contradiction avec l'utilisation de l'algorithme de Grover.**

La contradiction réside dans le fait que pour construire cet oracle, nous devons déjà avoir résolu le problème et si nous avons déjà résolu le problème, l'algorithme de Grover ne sert plus à rien. En effet, le but de Grover est de trouver une/des solutions inconnues à un problème en utilisant moins d'étapes qu'un ordinateur classique. Pour cela, l'algorithme de Grover utilise un Oracle qui renvoie "Vrai" uniquement pour les entrées solutions du problème. On ne connaît donc pas ses solutions préalablement et l'objectif de Grover est justement de trouver ses solutions. Or la méthode décrite en question 13 nécessite de construire la liste $L$ des solutions du problème pour construire l'oracle et ensuite appliquer l'algorithme de Grover : on utilise donc l'algorithme de Grover pour retrouver des solutions que nous avons déjà trouvées (et que nous venons de lui donner explicitement pour que celui-ci fonctionne).

**17. Proposer de façon schématique le fonctionnement de l'algorithme d'optimisation**

En utilisant, le circuit implémentant l'algorithme de Grover pour le problème de

décision, on peut arrivé assez facilement à la résolution du problème Max XOR SAT. En effet, dans le problème de décision, on fixe un entier $k$ et on vérifie s'il existe une solution satisfaisant au moins $k$ contrainte de l'instance du problème et exhibe cette solution le cas échéant. Le problème Max XOR SAT vise a trouvé la solution maximisant le nombre de contrainte satisfaite. Il suffit donc de trouver le $k$ le plus grand possible pour lequel le problème de décision trouve une solution.

En pratique, il suffit de faire commencer le circuit résolvant le problème de décision avec un $k = m$, si on ne trouve pas de solution, on ré-applique le circuit en baissant $k$ de 1 à chaque itération jusqu'à ce que le circuit trouve une solution, cette solution sera ainsi la solution maximisant le nombre de contraintes.

# 5 Évaluation des algorithmes

### Résultats Théoriques Attendus

On peut dans un premier temps évaluer la qualité théorique des 2 algorithmes que nous avons implémenter.

### QAOA

L'algorithme QAOA fait une alternance des opérateurs $e^{-i\gamma_j H_p}$ et $e^{-i\gamma_j H_{mixer}}$ un nombre $p$ fois (noté *reps* dans le code). On peut ainsi directement déduire que la profondeur du circuit associé est en $O(p \times profondeur(H_c) \times profondeur(H_{mixeur}))$.

Or d'après les expressions obtenues en question 7 et 8, on sait que la profondeur de l'opérateur de coût est de l'ordre de $O(m)$. De plus, on sait aussi que $H_{mixeur} = \sum_{j=1}^{n} X_j$, il n'a pas d'interaction entre les qubits, on peut donc appliquer les $X_j$ simultanément sur tous les qubits : la profondeur de cette opérateur est donc constante $O(1)$.

Finalement, la profondeur du circuit de l'algorithme de QAOA est de l'ordre de $O(p \times m)$.

De plus, le circuit n'a pas besoin de qubits ancillaires pour fonctionner. Il ne nécessite donc que n qubits (nombre de variables du problème).

Il est aussi à noter que la complexité réelle de l'algorithme dépendra également de la complexité du solveur classique utilisé pour optimiser les paramètres $(\gamma, \beta)$.

Enfin, on peut espérer que plus notre $p$ est grand plus notre algorithme permettra d'obtenir de meilleurs résultats (convergence vers le théorème adiabatique) au détriment du temps d'exécution qui évolue, à peu près, linéairement en fonction de $p$.

*Remarque :* Il existe probablement un nombre $p_{opti}$ qui permet d'obtenir la solution optimale avec une probabilité supérieur à un certain $\epsilon$ qu'on définit mais nous n'avons pas eu le temps de faire cette étude. Pour nos expériences, nous avons utilisé $p = 2$ car c'était un bon compromis entre bon résultat et temps d'exécution.

**Grover**

Pour résoudre le problème Max XOR SAT en utilisant l'algorithme de Grover, on résout le problème de décision associé avec Grover un nombre $(m - k_{opti})$ fois. Si on suppose que la profondeur de l'oracle est à peu près constante, on sait que la complexité d'une de ces résolutions est de l'ordre de $O(\sqrt{2^n})$ (nombre d'itérations optimale $= \frac{\pi}{4}\sqrt{2^n}$). Si on suppose, naïvement, qu'il en faut à peu près $\frac{m}{2}$ avant de trouver une solution, on peut conclure que la complexité totale de cette algorithme est de l'ordre $O(m\sqrt{2^n})$.

Pour la profondeur du circuit associé, étant donné que nous avons "triché" (cf. question 13) en construisant les listes des solutions $L_k$ au préalable pour chaque itération de k, les oracles ne sont pas représentatifs de l'oracle que nous aurions dû avoir sans "tricher" (cf. question 11). En effet, les oracles implémentés sont uniquement de taille $O(|L_k|)$ là où on peut imaginer qu'un oracle sans tricher, qui traduirait vraiment les toutes opérations arithmétiques du systèmes $Ax = b$, serait bien plus profond et complexe à construire. Il faut donc bien prendre cela en compte lors de l'analyse de nos mesures.

De la même manière, notre "tricherie" a permis de s'affranchir de qubits de travail (ancillas) nécessaire pour la construction de l'oracle comme on peut le voir en question 11 et 12, ce qui nous a permis de n'avoir besoin que de $n$ qubits avec l'oracle triché. Pour un oracle sans triche, il faudrait ajouter $m + q$ qubits de travail (1 pour chaque contrainte et $q$ le nombre de qubits nécessaire pour construire $I_n$) d'où un nombre total de qubits nécessaires de $n + m + q$.

Enfin, à l'issue des itérations pour trouver le bon $k_{opti}$, on applique juste une fois Grover pour le problème de décision, on devrait donc avoir une probabilité supérieur à $1 - \frac{4}{2^n}$ de mesurer la solution optimale.

**Résultats Expérimentaux**

*Remarques:*

1. Dans la résolution par Grover, pour éviter que le calcul des $L_k$ (brute-force) n'écrase les itérations de l'algorithme en terme de complexité, on ne mesurera que les calculs quantiques réalisés.
2. Pour l'algorithme de QAOA, bien entendu le temps d'exécution de l'algorithme et la qualité des solutions dépendra du nombre de *reps* (on utilise $reps = 2$ ici).

3. Qiskit étant un **simulateur** de machine quantique, l'exécution de QAOA et Grover sur celui-ci ne permet pas de battre la méthode énumération en terme de temps d'exécution (on simule les résultats qu'on obtiendrait sur un ordinateur quantique mais la simulation en elle même est classique). Toutefois, on peut comparer les temps d'exécution de QAOA et Grover entre eux étant donné qu'ils sont tout deux exécutés via qiskit.

### Analyse du temps d'exécution

On génère un jeu de données de manière aléatoire (des instances de taille allant de (2, 2) à (9, 9)) et on applique nos algorithmes dessus tout en mesurant le temps d'exécution de chacun d'eux.

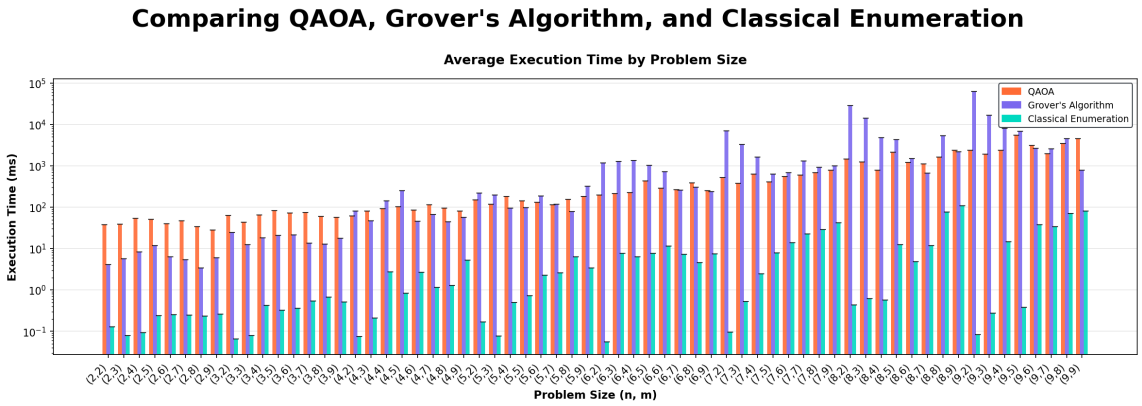



Figure 1: Benchmark comparant les performances de QAOA, Grover, Classique

Les résultats obtenus dans la Figure 1 sont cohérents avec ceux prédit théoriquement. En effet, on observe que pour des petites instances du problèmes, Grover à l'air plus efficace. On explique cela par les coûts fixes de la boucle d'optimisation classique de QAOA qui ralentissent inévitablement même pour des instances "petites", là où Grover lui n'a que $\frac{\pi}{4}\sqrt{2^n}$ itérations à faire (ce qui est peu pour des instances petites). Par contre, lorsque la taille des instances augmentent, les coûts fixes de la boucle d'optimisation de QAOA deviennent négligeables et on compare alors $O(p \times m)$ et $O(m\sqrt{2^n})$. Dès lors, bien que la racine de Grover permet une accélération par rapport au classique, le terme $m\sqrt{2^n}$ prend bien le dessus par rapport au facteur linéaire de QAOA pour des grandes instances : c'est donc QAOA qui gagne ici. (sans compter en plus le fait qu'on a un oracle "triché" pour Grover ici).

### Analyse de la qualité des solutions

Pour analyser la qualité de nos solutions (est-ce que nos algos ont trouvé un vecteur solution maximisant le nombre de contraintes satisfaites), on va les comparer avec l'algorithme exact (brute-force). On peut aussi noté qu'il est inutile de comparer si les algorithmes renvoient exactement le même vecteur solution : en effet, il peut exister plusieurs vecteurs solutions maximisant le même nombre de contrainte.

Pour cela, de la même manière qu'on a mesuré le temps d'exécution, on a généré un jeu de données aléatoirement (de taille (2, 2) à (9, 9)) sur lequel on applique nos algorithmes.

On regroupe dans le Tableau 1 un extrait des performances de nos algorithmes.

Table 1: Comparaison d'optimalité : Bruteforce (Référence), Grover et QAOA

| **Instance** $(n, m)$ | **Max Théorique** | **Grover** | **QAOA** |
|---|---|---|---|
| (2, 2) | **2** | **2** | **2** |
| (2, 9) | **7** | **7** | **7** |
| (5, 4) | **4** | **4** | **4** |
| (5, 9) | **8** | **8** | 5 |
| (6, 9) | **7** | **7** | **7** |
| (8, 8) | **8** | **8** | 6 |
| (8, 9) | **8** | **8** | 6 |
| (9, 2) | **2** | **2** | **2** |
| (9, 9) | **8** | **8** | **8** |

*Note : QAOA est exécuté avec une profondeur $p = 2$.*

Sur l'intégralité du jeu de données, l'algorithme de Grover a trouvé la solution optimale dans **100%** des cas. Cela confirme la nature "exacte" de l'algorithme. En effet, la stratégie d'utiliser le problème de décision avec un $k = m$ qu'on diminue au fur et à mesure garantit qu'à un moment, on trouve une solution et que celle-ci est de poids maximum : l'oracle ne peut pas "louper" la solution si elle existe, et amplifie donc forcément son amplitude nous permettant de la mesurer, pour peu que le nombre d'itérations soit calibré correctement. Finalement, seule la mesure finale est probabiliste (qu'on peut contrer en multipliant le nombre de mesure), l'oracle lui est déterministe.

L'algorithme QAOA a trouvé la solution optimale dans la majorité des cas mais montré ses limites pour des instances où le nombre de contraintes est élevés par rapport au nombre de variables : on mesure 17 erreurs sur 64 i.e **74%** de réussite. Ces échecs ne sont pas étonnant pour une profondeur $p = 2$. En effet, l'algorithme

de QAOA est une heuristique ce qui fait que sur certaines instances difficiles du problème, on peut se retrouver "bloqué" sur des optimums locaux. En refaisant la même simulation mais pour un $p = 5$, on obtient un taux de réussite bien meilleur à **89%**.

On note quand même que même quand l'algorithme se trompe, il nous renvoie une solution "acceptable" relativement proche du maximum théorique.

### Conclusion

A travers ces expériences, on peut arriver à la conclusion que l'algorithme de Grover garantie une optimalité des solutions et une accélération quadratique par rapport au classique mais sa complexité d'implémentation et d'exécution (surtout avec un oracle sans tricher) le rend inadapté pour des instances qui commence à être significatives. De l'autre côté QAOA nous offre une solution plus pragmatique, qui certe ne garantie pas l'optimalité des solutions, mais possède une complexité temporelle seulement linéaire par rapport à la taille du problème et la précision qu'on désire, le rendant utilisable en pratique.