# Projet : résolution quantique de MAX XOR-SAT

UE Informatique quantique et Recherche opérationnelle

7 janvier 2025

ensIIE - MPRO

Ce TP noté se fera en quadrinôme, et devra être restitué (code + réponses aux questions) en au plus tard **le 18 janvier 2026 à 23h59** sur `exam.ensiie.fr` dans le dépôt `iqro-tp-note`. Le temps imparti encadré est de trois fois 1h45, soit 4h45 au total.

Il est conseillé pour les personnes n'ayant pas suivi de cours de recherche opérationnelle de se mettre avec une personne ayant suivi un tel cours. Les élèves du MPRO peuvent se mettre ensemble ou non.

Le but de ce TP est de résoudre le problème max XOR SAT : QAOA et l'algorithme de Grover. Il s'agit d'une variante de la programmation linéaire en nombre entier avec des contraintes de parité.

**Les codes devront être rendus dans des fichiers `.py`; tous dans un unique dossier `code`. Des fichiers jupyter ou équivalents ne sont pas attendus. Les questions théoriques doivent être rendues dans un dossier `questions` dans un ou plusieurs fichiers pdf (latex compilé ou odt exporté) ou des images scannées d'une page manuscrite (de préférence de taille maîtrisée). Indiquer dans le code et les rapports le numéro de la question à laquelle vous répondez.**

## 1 Description du problème et question préliminaire

On considère une matrice $A \in M_{m,n}\{0,1\}$ et un vecteur $b \in \{0,1\}^m$. Trouver un vecteur $x \in \{0,1\}^n$ maximisant le nombre d'égalités satisfaites dans le système suivant:

$$Ax = b \mod 2$$

Par exemple si $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ et $b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, on a le système suivant:

$$\begin{aligned} x_1 + x_2 &= 0 \mod 2 \\ x_2 &= 1 \mod 2 \end{aligned}$$

Dit autrement, $x_1 + x_2$ est pair et $x_2$ est impair. On peut résoudre les deux équations du système avec $x_1 = x_2 = 1$. Parfois le système ne peut être résolu et on cherche dans ce cas à maximiser le nombre d'égalités qui sont vraies.

Ce problème est NP-Difficile (on ne connait pas d'algorithme polynomial pour le résoudre et il y a peu de chances que cela arrive).

1. Ecrire une classe MaxXorSat qui représente une entrée du problème.

2. Ecrire une fonction qui prend en entrée une instance du problème MaxXorSat et résout l'instance. La réponse de la fonction doit être la solution et son utilité.

   *Vous pouvez utiliser un algorithme d'énumération de toutes les solutions, ou un programme linéaire - vous trouverez une formulation linéaire dans la section 5.2.2 de* `https://arxiv.org/abs/1309.6827`.

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

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

## 2 Résolution avec QAOA

QAOA nécessite en entrée un problème sous forme HOBO (High Order Binary Optimization), c'est-à-dire minimiser une fonction polynomiale à variables binaires sans contrainte.

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

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

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 satisfaites.

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

   On va maintenant pouvoir s'attaquer au code.

   **Conseil pour coder QAOA avec Qiskit:**

   - Si vous avez bien compris l'algorithme, vous devriez pouvoir coder vous même l'algorithme. Mais ce serait un peu laborieux. Qiskit vous propose une boîte noire toute faite pour cela. Cette boîte noire nécessaire malgré tout l'hamiltonien.
   - Vous pouvez décrire un hamiltonien très facilement avec Qiskit. On peut encoder une porte $A_1 \otimes A_2 \otimes \cdots \otimes A_n$ où $A_i \in \{Z, I\}$ avec la chaîne de caractères $A_n A_{n-1} \cdots A_2 A_1$ (où $A_i$ est remplacé par le caractère `Z` ou `I` selon sa valeur). Attention à l'ordre des portes!

     Par exemple la porte $Z \otimes Z \otimes I$ pourra être encodée avec la chaîne `IZZ`.

     Ensuite, un hamiltonien $H_C = \alpha \cdot A_1 \otimes A_2 \otimes \cdots \otimes A_n + \beta B_1 \otimes B_2 \otimes \cdots \otimes B_n + \cdots + \gamma C_1 \otimes C_2 \otimes \cdots \otimes C_n$ pourra être construit avec la fonction suivante:
     `SparsePauliOp.from_list([("`$A_n A_{n-1} \cdots A_1$`", `$\alpha$`), ("`$B_n B_{n-1} \cdots B_1$`", `$\beta$`), ..., ("`$C_n C_{n-1} \cdots C_1$`", `$\gamma$`)])`

     Par exemple $H_C = Z \otimes Z \otimes I + 2I \otimes Z \otimes I$ pourra être créé avec
     `SparsePauliOp.from_list([("IZZ", 1), ("IZI", 2)])`
   - Une fois l'hamiltonien connu, vous pouvez construire le circuit de QAOA avec l'opérateur `QAOAAnsatz` qui prend en entrée l'hamiltonien et un paramètres `reps` égal au nombre de répétition des alternances des exponentielles des portes $H_C$ et $H_B$.
   - Rappelez vous que QAOA construit un circuit paramétré, il faut, pour le faire fonctionner donner une valeur à ses paramètres. Il faut ensuite trouver le bon set de paramètres pour que ce circuit vous donne la meilleure solution. Vous devez donc optimiser les paramètres du circuit avec un algorithme classique. Vous pouvez utiliser un optimiseur déjà conçu pour ça.

     Commencez par récupérer le nombre de paramètres du circuit:
     `q.num_parameters`

     Fixez ensuite ces paramèters à des valeurs aléatoires, par exemple entre 0 et $2\pi$.

     Il nous faut une fonction capable d'évaluer un set de paramètres `params`. On utilisera la fonction suivante (à adapter en fonction de la version de qiskit que vous utilisez):

```
1  from qiskit.primitives import StatevectorEstimator
2
3  def f(params):
4      """
5      On suppose qu'on a deux variables globales.
6      q est le circuit paramétré renvoyé par QAOAAnsatz
7      hamiltonian est l'hamiltonien généré avec la fonction SparsePauliOp.from_list
8      """
```

```
9
10      pub = [q, [hamiltonian], [params]]
11      estimator = StatevectorEstimator()
12      result = estimator.run(pubs=[pub]).result()
13      cost = result[0].data.evs[0]
14
15      return cost
```

Ce circuit évalue la qualité des paramètres `params` au travers de l'hamiltonien. Pour simplifier grossièrement, la fonction `estimator.run` effectue les opérations suivantes:

- Instancier les paramètres du circuit `q` avec les valeurs données dans la liste `params`
- Exécuter le circuit `q` de nombreuses fois pour produire une distribution de qbits
- Tester chaque vecteur de cette distribution sur l'hamiltonien pour déterminer la valeur propre moyenne sur cette distribution. Plus la valeur propre moyenne est petite, plus le circuit a de grandes chances de sortir une solution réalisable de bonne qualité.

*Plus d'informaton dans le premier TP sur qiskit.*

On peut ensuite utiliser ensuite la fonction de `scipy` nommée `minimize` pour trouver un bon set de paramètres.

```
best_params = minimize(f, init_params, args=(), method="COBYLA").x
```

Enfin, on peut rappeler `f` avec ces paramètres pour avoir la valeur objective finale ou mesurer manuellement la distribution finale et extraire la meilleure solution. Pour instancier manuellement les paramètres, il faut utiliser:

```
q.assign_parameters
```

9. Ecrire un code, commenté, résolvant le problème de max XOR SAT avec l'algorithme QAOA.

# 3 Résolution avec l'algorithme de Grover

Afin de résoudre le problème max XOR SAT avec l'algorithme de Grover, nous allons commencer par résoudre le problème de décision associé.

## 3.1 Résolution du problème de décision associé

L'oracle de l'algorithme de Grover associé au problème de *décision* de max XOR SAT s se décompose en deux parties. La première partie marque uniquement les solutions réalisables, tandis que la seconde partie sélectionne les solutions dont la valeur est supérieure à un entier fixé $k$.

10. Écrire le problème de décision associé au 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.

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).

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ et } b = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

13. Pour ne pas exploser le nombre de qbits avec les qbits de travail dans qiskit en construisant la porte $In$, 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 qbit de base $|\underline{x}\rangle$ renvoie $-|\underline{x}\rangle$ si $x \in L$ et renvoie $|\underline{x}\rangle$ sinon.

14. Implanter ensuite une fonction qui commence par construire $L$ la liste de toutes les solutions réalisables dont le poids est supérieur à $k$. Utiliser ensuite cette fonction pour construire un oracle.

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

16. Codez (malgré la contradiction) avec qiskit un circuit qui implante l'algorithme de Grover pour résoudre le problème de décision associé au problème max XOR SAT.

### 3.2 Résolution du problème d'optimisation

Nous revenons au problème initial, qui est le problème d'optimisation max XOR SAT.

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

18. Implémenter cet algorithme avec Qiskit.

## 4 Evaluation

Proposer une évaluation de la qualité des 2 algorithmes, QAOA et Grover. Cet algorithme évaluera:

- le temps de calcul
- la qualité des solutions renvoyées, utilisez pour cela l'algorithme exact codé au début du projet.
- la complexité des algorithmes (nombre de porte, profondeur des circuits, nombre de qbits, nombre de répétition des algorithmes quantiques).