IQRO

Sylwyn Giardi & Hugo Lallemant & Mickaël Saes-Vincensini

January 2025

1 Description du problème et question préliminaire

1.1 Question 1

Soit $x_i \in \{0,1\}$, valant 0 si l'objet n'est pas dans le sac et 1 sinon. La modélisation mathématique du problème est :

$$max_{x\in\{0,1\}}(\sum_{i=1}^{n} u_i x_i) \tag{1}$$

$$\sum_{i=1}^{n} v_i x_i = V \tag{2}$$

1.2 Question 2 et 3

Réponse dans le fichier knapsack.py.

2 Résolution avec QAOA

2.1 Question 4

Soit

$$g(x) = \sum_{i=1}^{n} v_i x_i - V$$

On se donne la contrainte C :

$$g(x) = 0$$

La contrainte est vérifiée uniquement quand le sac à dos est plein.
Posons maintenant $p_c(x) = (g(x))^2$
On a bien p_c qui vérifie les conditions suivantes :

$$\begin{cases} p_c(x) = 0 \leftrightarrow g(x) = 0 \\ p_c(x) > 0 \leftrightarrow g(x) \neq 0 \end{cases} \tag{3}$$

On pose $p_c : x \to (g(x))^2$ et $\lambda_c \in \mathbb{R}^+$.
On cherche finalement à répondre au problème suivant :

$$min_{x\in\{0,1\}}(\lambda_c p_c(x) - \sum_{i=1}^{n} u_i x_i) \tag{4}$$

2.2 Question 5

On fait le changement de variable suivant $z_i = 2 * x_i - 1$, on a bien $z_i \in \{-1, 1\}$ et on résout le problème suivant :

$$min_{z\in\{-1,1\}}(\lambda_c(\sum_{i=1}^{n} v_i \frac{z_i+1}{2} - V)^2 - \sum_{i=1}^{n} u_i \frac{z_i+1}{2}) \tag{5}$$

2.3 Question 6

$$H_c = \lambda_c(\sum_{i=1}^{n} v_i \frac{Z_i + I_{2^n}}{2} - V * I_{2^n})^2 - \sum_{i=1}^{n} u_i \frac{Z_i + I_{2^n}}{2}$$

Avec Z_i, matrice carrée de taille 2^n correspondant au circuit ayant la porte Z appliquée au qbit i.
On développe ce calcul pour le mettre sous forme du modèle d'Ising :

$$H_c = \lambda_c(\frac{1}{2}\sum_{i=1}^{n} v_i Z_i + \frac{1}{2}\sum_{i=1}^{n} v_i I_{2^n} - V * I_{2^n})^2 - \frac{1}{2}\sum_{i=1}^{n} u_i Z_i - \frac{1}{2}\sum_{i=1}^{n} u_i I_{2^n}$$

On pose $\sum_{i=1}^{n} v_i = S_v$ et $\sum_{i=1}^{n} u_i = S_u$

$$H_c = \lambda_c(\frac{1}{2}\sum_{i=1}^{n} v_i Z_i + (\frac{1}{2}S_v - V) * I_{2^n})^2 - \frac{1}{2}\sum_{i=1}^{n} u_i Z_i - \frac{1}{2}S_u * I_{2^n}$$

$$H_c = \lambda_c(\frac{1}{4}(\sum_{i=1}^{n} v_i Z_i)^2 + (\frac{1}{2}S_v - V)\sum_{i=1}^{n} v_i Z_i + (\frac{1}{4}S_v^2 - VS_v + V^2) * I_{2^n}) - \frac{1}{2}\sum_{i=1}^{n} u_i Z_i - \frac{1}{2}S_u * I_{2^n}$$

$$(\sum_{i=1}^{n} v_i Z_i)^2 = \sum_{i=1}^{n} (v_i Z_i)^2 + 2\sum_{i<j} v_i v_j Z_i Z_j = \sum_{i=1}^{n} v_i^2 I_{2^n} + 2\sum_{i<j} v_i v_j Z_i Z_j$$

(car Z_i est diagonale et donc commute et $Z_i^2 = I_{2^n}$)
On pose $\sum_{i=1}^{n} v_i^2 = S_{v2}$
En regroupant les termes linéaires et bilinéaires on obtient donc :

$$H_c = \sum_{i=1}^{n} (\lambda_c(\frac{1}{2}S_v - V)v_i - \frac{1}{2}u_i)Z_i + \frac{\lambda_c}{2}\sum_{i<j} v_i v_j Z_i Z_j + \lambda_c((\frac{1}{4}(S_{v2} + S_v^2) - VS_v + V^2) - \frac{1}{2}S_u) * I_{2^n}$$

$$H_c = \sum_{i=1}^{n} h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j + C * I_{2^n}$$

Avec :
— $h_i = \lambda_c(\frac{1}{2}S_v - V)v_i - \frac{1}{2}u_i$
— $J_{ij} = \frac{\lambda_c}{2} v_i v_j$
— $C = \lambda_c((\frac{1}{4}(S_{v2} + S_v^2) - VS_v + V^2) - \frac{1}{2}S_u$

L'Hamiltonien est testé dans le fichier hamilton.py, avec l'exemple V=33, Volumes=[10, 12, 11, 30], Utilités=[11,21,10,100] la solution est l'utilisation des trois premiers objets et donc -42 doit être valeur propre de l'Hamiltonien calculé ce qui est bien le cas.

2.4 Question 8

Réponse dans le fichier QAOA.py

3 Partie 3

3.0.1 Question 9

On représente le problème du sac à dos sous la forme d'un problème de décision.
Étant donné :
— Un volume maximal V
— Un ensemble d'objets avec leurs volumes respectifs $v_1, v_2, \ldots, v_n$
— Les utilités associées $u_1, u_2, \ldots, u_n$
— Une valeur cible k

Le problème de décision consiste à répondre à la question suivante :

Existe-t-il un sous-ensemble d'objets dont la somme des volumes est égale à V et dont la somme des utilités est supérieure ou égale à k ?

3.0.2 Question 10

L'oracle sera découpé en deux parties, de la même manière que le problème de décision. La première partie de l'oracle permettra de lister toutes les solutions réalisables, ici le fait que la somme des volumes = volume max.

Dans la deuxième partie, on sélectionne les solutions dont la somme des utilités dépasse une certaine valeur k. Si il y a une seule solution, on a la solution optimale.

Si il y en a plusieurs, il faut augmenter k, si il n'y en a pas, il faut diminuer k.

3.0.3 Question 11

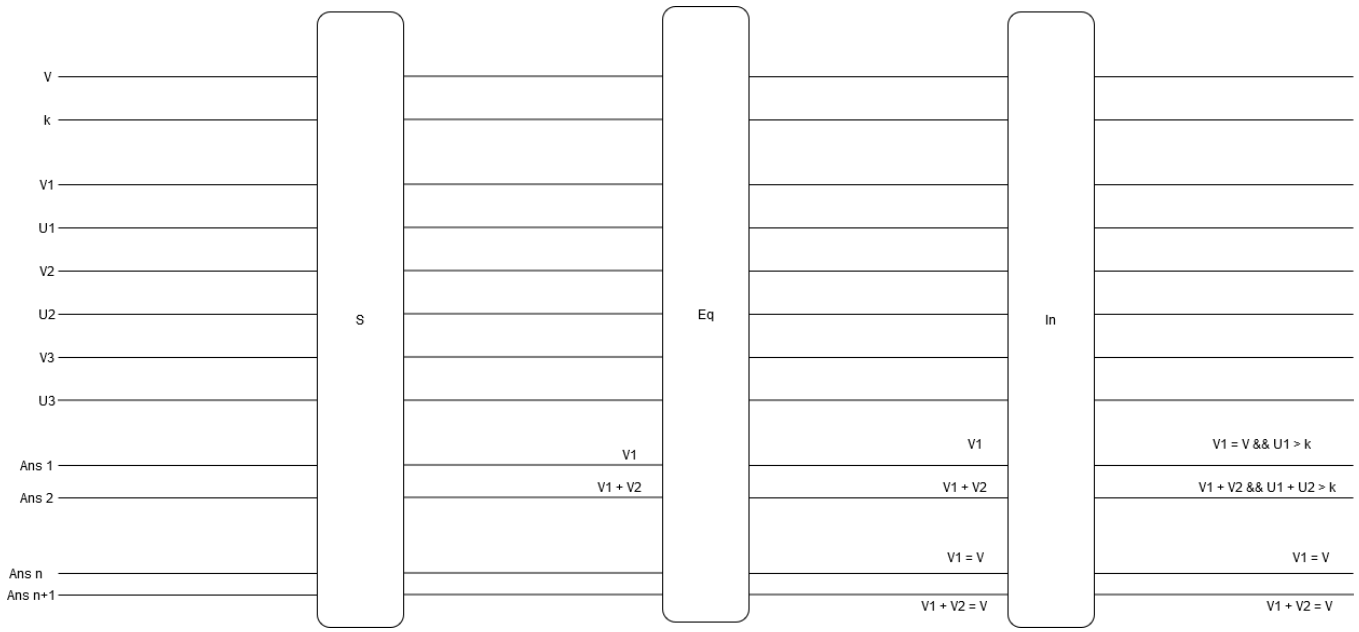

FIGURE 1 – *Schéma du circuit quantique*

Dans un premier temps, avec la porte S, on va lire en entrée V1, V2 et V3 pour sortir toutes les combinaisons possibles des sommes de ces trois nombres. De même pour U1,U2 et U3.
On va ensuite avec la porte Eq vérifier si les sommes correspondent à la valeur V.
Avec la porte In, on va vérifier si la sommes de utilités est supérieure à la valeur k et si la somme des volumes est bien de V.

3.0.4 Question 12

Soit L_i un élément de L. On se donne la logique suivante pour renvoyer x si $x = L_i$, on va ensuite enchaîner ces tests sur tous les éléments de L.

On se donne la représentation binaire de L_i, on va supposer $L_i \neq 0$ sinon si x vaut 0, c'est son propre inverse.

Si $L_i = 011$ en représentation binaire, on place des portes X sur tous les emplacements où la représentation binaire contient un 0, c'est-à-dire pour l'exemple on place une porte X sur le premier qubit de x. Si $L_i = 001$, on aurait placé une porte X sur le premier et second qubit de x.

Ensuite, on place une porte Z contrôlée sur un des emplacements où L_i vaut 1, pour $L_i = 011$, on pourrait placer une porte Z contrôlée sur le deuxième qubit, contrôlé par les deux autres qubits de x.

On place ensuite une autre rangée de portes X pour inverser l'effet initial.

On fait tout cela car, si x est effectivement égal à L_i, alors après la première rangée de portes X, on aura le qubit 111 par exemple. Ensuite, la porte Z contrôlée renverra -111, et les portes X finales permettent d'avoir -x, c'est-à-dire ce que l'on souhaite.

Si x n'est pas égal à L_i, alors, après les portes X, le qubit ne sera pas égal à 111, alors la porte z contrôlée n'aura aucun effet :

— Si le qubit sur le quel on applique la porte Z contrôlée vaut 0, alors Z appliqué sur 0 renvoie 0, on inverse pas.
— Si c'est un qubit de contrôle qui vaut 0, alors la porte Z ne s'applique pas, on inverse pas.

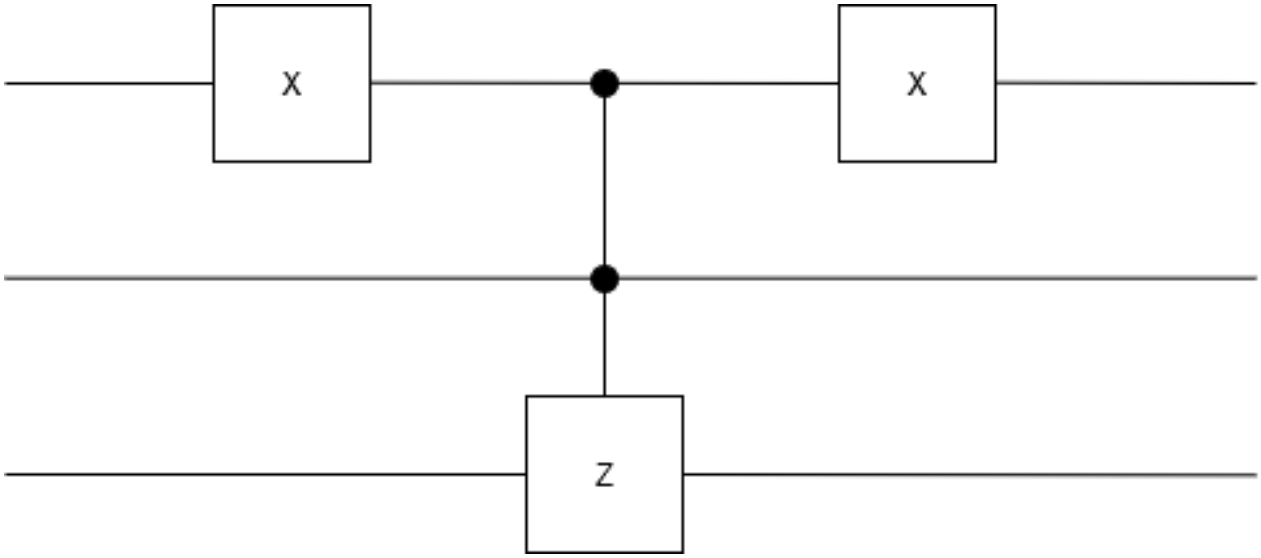

FIGURE 2 – Exemple dans le cas où $L_i = 011$.

Finalement, on va appliquer ça à la chaîne, comme x est unique, s'il est dans L, on va l'inverser une seule fois. Ce circuit a bien une profondeur finale de O(|L|), car la profondeur est de $3 \times |L|$

3.0.5 Question 13

L'énoncé de la question comprenait : "L la liste de toutes les solutions réalisables dont le poids est supérieur à k".

Nous avons supposé qu'il fallait le corriger en "L la liste de toutes les solutions réalisables dont **l'utilité** est supérieur à k" car cela faisait plus de sens selon nous.

Nous n'avons pas eu l'impression que cela changeait le résultat sur les exemples qui suivent. La fonction peut se modifier assez simplement pour s'adapter de toute manière.

3.0.6 Question 14

La contradiction avec Grover repose dans le fait que l'algorithme de Grover permet de trouver les solutions d'un tel problème. Dans notre cas, nous générons à l'avance la liste L des cas possibles. Cela est donc une contradiction avec Grover car on "casse le principe de l'algorithme".

3.0.7 Question 15

Réponse dans le fichier grover.py

Le code de cette question est partiellement fait et ne fonctionne pas. Les probabilités de sorties semblent trop faibles quand on met la solution optimale.

3.1 Question 16

L'algorithme d'optimisation a le fonctionnement suivant :

— On définit les paramètres d'entrée et on les initialise
— On construit l'algorithme d'optimisation de la sorte
  — On place une rangée de portes H

— On calcule le nombre d'itérations à effectuer comme $\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor$ où N est le nombre total d'états possibles (qui vaut 2^n), et M le nombre de solutions qui vont satisfaire les contraintes, c'est-à-dire qui ont un volume égal au volume total et une utilité supérieure à une valeur k
— On applique l'oracle et la diffusion ce nombre de fois
— On mesure les qubit, et on interprète comme : Si on a 101, on sélectionne les objets d'indice 0 et 2, et on ne prend pas celui d'indice 1.

3.2 Question 17

Réponse dans le fichier grover.py

4 Partie 4

Un fichier de test est présent dans *code/test.py*

On remarque ainsi que grover nous donne le bon résultat très rapidement, vérifié par un algorithme dynamique utilisé en début de projet.
Cependant l'algorithme QAOA prends un temps très important et retourne des fois le bon résultat, mais ce n'est pas fiable à chaque utilisation. Pour avoir un meilleur résultat, il faut faire un nombre très important d'itérations.