

Generación de bases mutuamente no sesgadas en Qiskit

Juan M. Pujol^{1,2}, Mariela Portesi^{1,2}, and Federico Holik¹

¹ Instituto de Física de La Plata, Argentina
pujol@iflp.unlp.edu.ar,

² Universidad Nacional de La Plata, Argentina

Abstract. Las bases mutuamente no sesgadas (MUBs, por sus siglas en inglés) son conjuntos de bases que representan la noción de complementariedad. Estas estructuras son ampliamente estudiadas por su relevancia en diversas áreas, abarcando desde la matemática pura hasta aplicaciones en información cuántica, como distribución cuántica de claves (QKD) y tomografía de estados cuánticos. Un conjunto completo de MUBs en dimensión d consiste en $d + 1$ bases, y se sabe que existe únicamente cuando d es un número primo o potencia de un primo. En este trabajo implementamos un algoritmo en Qiskit para calcular el conjunto completo de bases mutuamente no sesgadas en sistemas de n qubits (dimensión 2^n), siguiendo el circuito presentado en Yu and Dongsheng, 2023. Esta contribución proporciona una herramienta práctica para aplicaciones en computación cuántica, permitiendo la generación programática de estas bases con énfasis en escalabilidad y verificación de propiedades teóricas.

Keywords: MUBs, Qiskit, medidas, computación, circuito

Computing mutually unbiased bases in Qiskit

Juan M. Pujol^{1,2}, Mariela Portesi^{1,2}, and Federico Holik¹

¹ Instituto de Física de La Plata, Argentina
pujol@iflp.unlp.edu.ar,

² Universidad Nacional de La Plata, Argentina

Abstract. Mutually Unbiased Bases (MUBs) are sets of bases which represent the notion of complementarity, central in quantum mechanics. These structures are widely studied due to their relevance in a number of areas, ranging from pure mathematics to quantum information applications, such as quantum key distribution, and quantum state tomography. A full set of MUBs in d dimensions consists in $d + 1$ bases, and it is known to exist only when d is a prime number or a prime power.

In this work, we implement an algorithm in Qiskit to compute the full set of mutually unbiased bases for systems of n qubits (2^n dimensions), following the circuit presented in Yu and Dongsheng, 2023. This contribution provides a practical tool for applications in quantum computing, allowing a systematic generation of these bases with an emphasis on scalability.

Keywords: MUBs, Qiskit, measurement, computing, circuit

1 Introducción

En mecánica cuántica, el proceso de medición es de importancia fundamental. Escoger un conjunto de bases adecuadas para la extraer la información deseada de la medida es entonces un problema que requiere de especial cuidado, generando así un amplio campo de estudio.

En este contexto, las **Bases Mutuamente no Sesgadas**, o MUBs por sus siglas en inglés, emergen como un conjunto de particular interés. Dado un espacio de Hilbert de dimensión d , dos bases ortonormales $\{|e_i\rangle\}$ y $\{|f_j\rangle\}$ son mutuamente no sesgadas si el valor absoluto del producto interno entre un vector de una base y cualquier vector de otra base es constante, es decir, $|\langle e_i|f_j\rangle| = \frac{1}{\sqrt{d}}$ para todo i, j . Físicamente, esto implica que una medida en una de las bases implica total desinformación sobre una medida en otra de las bases, representando una noción de complementariedad.

Las MUBs son de gran interés tanto teórico como práctico, ya que sus propiedades conllevan una gran utilidad en ciertas aplicaciones. Como ejemplo, sirven como bases óptimas para realizar tomografía de estados cuánticos (Adamson and Steinberg, 2010), implementación de protocolos de distribución cuántica de claves (Ikuta, 2022), y códigos de corrección de errores (Ryan-Anderson, 2021).

A pesar de lo mencionado, muchos problemas con respecto a su existencia y construcción permanecen abiertos (Durt, 2010). Si bien es sabido que un conjunto completo de $d + 1$ MUBs existe en \mathbb{C}^d cuando d es primo o potencia de primo, tanto la existencia como la construcción de conjuntos completos en otras dimensiones son problemas sin resolver a día de hoy. De hecho, aún para dimensiones donde su existencia y construcción teórica está demostrada, su confección a partir de dicha construcción es un problema nada trivial en si mismo.

En este contexto, resulta de particular interés un trabajo de Yu, W. Dongsheng, W. (2023), donde se presenta un algoritmo eficiente para la construcción del conjunto completo de MUBs para dimensiones de 2^n . En nuestro trabajo, implementamos el algoritmo en la plataforma de Qiskit, haciendo énfasis en la facilidad de uso para un usuario externo, y en la conveniencia para su aplicación en algoritmos cuánticos.

2 Circuito

Un circuito cuántico se puede identificar con una matriz unitaria U . Esta matriz representa la aplicación de todas las compuertas que forman el circuito, y actúa sobre todos los qubits. En nuestro caso particular, se pueden reconocer a grandes rasgos 4 fases del circuito: en primer lugar la preparación de los estados, luego una primera capa de operaciones mediante compuertas de Haddamard, una segunda capa de compuertas S , y una última de compuertas control- Z . La combinación particular de estas compuertas variará dependiendo de la dimensión, la base, y el vector de la misma que estemos preparando.

En lo consiguiente, denotaremos por n al número de qubits recordando que la dimensión vendrá dada por $d = 2^n$. Como fue mencionado, el conjunto completo de MUBs contiene $d + 1$ bases, de d vectores d -dimensionales cada uno. Si bien los fragmentos más relevantes del código serán explicados a continuación, se puede ver el código completo en el siguiente repositorio de Github: https://github.com/juanmpujol/MUB_generator_qiskit/

2.1 Primera fase: preparación del estado

Antes de aplicar las compuertas pertinentes, es necesario preparar el estado inicial; estos serán todos los estados posibles de la base computacional. Recordemos que los estados de esta base se representan como $|0 \dots 00\rangle$, $|0 \dots 01\rangle$, ..., $|10 \dots 1\rangle$, $|11 \dots 1\rangle$, lo cual no es más que todas posibles combinaciones de n ceros y unos. La aplicación de las compuertas al estado i -ésimo de la base computacional dará como resultado el estado i -ésimo de la MUB correspondiente.

Para generar el estado i -ésimo de la base computacional a partir del estado inicial $|0 \dots 0\rangle$ aplicamos una serie de compuertas X , cuyo efecto viene dado por $X|0\rangle = |1\rangle$. El extracto del código relevante se muestra a continuación:

```
qc = QuantumCircuit(n)

combs = np.array([set()] + [combo for i in range(1, nq + 1) for combo
    in combinations(range(nq), i)], dtype = object)
relevant_combs = combs[state_index]
for i in range(len(relevant_combs)):
    qc.x(relevant_combs[i])
```

Este fragmento, después de inicializar el circuito cuántico con la cantidad de qubits requerida, enumera todas las combinaciones posibles de compuertas X , representando todos los posibles estados de la base computacional; luego, dependiendo del índice del estado, aplica las correspondientes. Una vez aplicadas, el sistema se encuentra en el estado de la base computacional i -ésimo.

2.2 Segunda fase: compuertas H y S

Una vez generado el estado inicial, se le aplica a todos los qubits una compuerta de Haddamard H . Esta capa es igual para todos los estados en todas las bases.

Luego, se aplican las compuertas S necesarias. Si j es el índice de la base, la componente de la matriz $U(j)$ correspondiente a esta fase se puede representar como $U_S(j) = S^{a_0(j) \otimes \dots \otimes a_{n+1}(j)}$. Los índices $a_r(j)$ pueden tomar valores de $\{0, 1, 2, 3\}$, y su cálculo requiere de una serie de cálculos matemáticos sobre el campo de Galois $GF(2^n)$. En particular, los elementos sobre $GF(2^n)$ se pueden expresar en forma tanto vectorial como entera; las funciones encargadas del pasaje entre una representación y la otra se encuentran en el código:

```
def integer_form(x):
    return sum([x[i]*2**i for i in range(len(x)) ])

def vector_form(x):
    binary_list = [int(i) for i in bin(x)[2:]]
    binary_list.reverse()
    auxzeros = [0]*(nq-len(binary_list))
    binary_list.extend(auxzeros)
    return binary_list
```

Para obtener los valores de $a_r(j)$, se debe evaluar una dupla de productos en $GF(2^n)$, representado por el siguiente fragmento de código:

```
def a(j, r):
    a1 = (vector_form(j) @ (M_0 @ x[2*r].T)) % 2
    a2 = (vector_form(j) @ (M_1 @ x[2*r].T)) % 2
    if a1 == 0:
        return 0 if a2 == 0 else (2 if a2 == 1 else None)
    elif a1 == 1:
        return 3 if a2 == 0 else (1 if a2 == 1 else None)
    else:
        return None
```

donde \mathcal{M}_i y x^{2^r} son respectivamente matrices y vectores obtenidos a partir de los coeficientes de un polinomio irreducible de grado n , calculados en otro fragmento del código. Una vez obtenidos los $a_r(j)$, implementamos las matrices H y S de la siguiente manera:

```
for qubit in range(n):
    qc.h(qubit)
for i in range(a(j, qubit)):
    qc.s(qubit)
```

2.3 Tercera fase: compuertas CZ

La tercera y última capa de compuertas corresponde a una serie de control-Z siguiendo la fórmula: $UCZ(j) = \prod_{0 \leq s < t \leq n-1} CZ(s, t)^{b_{s,t}(j)}$, donde $CZ(s, t)$ es la operación CZ con el qubit de control q_s y el q_t como el objetivo. De una manera similar a los $a_r(j)$, los coeficientes $b_{s,t}(j)$ se calculan mediante la siguiente función:

```
def b(j, s, t):
    return (vector_form(j) @ (M_0 @ x[s+t].T)) % 2
```

Una vez calculados, se aplican las compuertas a los qubits que correspondiesen de la siguiente forma:

```
for s in range(nq):
    for t in range(s+1, nq):
        if b(j, s, t) == 1:
            qc.cz(s, t)
```

Una vez aplicadas estas compuertas solo resta obtener el estado final, lo cual se logra midiendo con la línea:

```
sv = quantum_info.Statevector.from_instruction(qc).data
```

El resultado final sv es entonces un vector perteneciente a la base j -ésima en 2^n dimensiones. Para obtener la base completa solo resta hacer el proceso adecuado para todos los vectores de la base computacional, y para obtener el conjunto completo de bases se recorren los valores de $j = (0, 1, \dots, 2^n - 1)$. Esto se hace con el siguiente fragmento:

```
def full_basis(nq, j):
    all_states = []
    for i in range(2**nq):
        all_states.append(get_state(nq, j, i))
    return np.array(all_states)
def full_set(nq):
    set_matrix = []
    set_matrix.append(np.eye(2**nq))
    for j in range(2**nq):
        set_matrix.append(full_basis(nq, j))
    return np.array(set_matrix)

fullset_array = full_set(nq)
```

Obteniendo así el conjunto completo de bases mutuamente no sesgada en la dimensión deseada.

3 Conclusión

En este trabajo presentamos una implementación en Qiskit de un algoritmo que permite reconstruir un conjunto de MUBs para un número arbitrario de qubits. Estos códigos constituyen implementación de la teoría desarrollada en Yu, W. & Dongsheng, W. (2023), y podrían encontrar aplicaciones en distintos problemas de teoría de la información cuántica, tales como estimación de estados cuánticos y detección de entrelazamiento. Además, una breve guía de uso se encuentra publicada en el repositorio de Github correspondiente, para facilitar su implementación dentro de algoritmos más complejos.

References

- Adamson, R. B. A., & Steinberg, A. M. (2010). Improving quantum state estimation with mutually unbiased bases. *Phys. Rev. Lett.*, *105*, 030406.
- Durt, T. e. a. (2010). On mutually unbiased bases. *International Journal of Quantum Information*, *08*(04), 535–640.
- Ikuta, T. e. a. (2022). Scalable implementation of $(d + 1)$ mutually unbiased bases for d -dimensional quantum key distribution. *Phys. Rev. Res.*, *4*, L042007.
- Ryan-Anderson, C. e. a. (2021). Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, *11*, 041058.
- Yu, W., & Dongsheng, W. (2023). An Efficient Quantum Circuit Construction Method for Mutually Unbiased Bases in n -Qubit Systems.