

Entropía de Shannon vs Entropía de von Neumann



POLITÉCNICA

UPM

Sergio Heras Álvarez
Ricardo Ferreiro de Aguiar

ÍNDICE

Introducción	3
Entropía de Shannon	3
Fundamentos teóricos.....	3
Ejemplo.....	5
Entropía de von Neumann	7
Fundamentos teóricos.....	7
Enunciado del Problema.....	8
Resolución del Problema.....	9
Explicación del Problema	10
Ampliación del Primer Trabajo.....	12
IBM	12
Microsoft	15
D-Wave.....	16

Introducción

En el campo de la información cuántica, hay dos conceptos fundamentales: la Entropía de Shannon y la Entropía de von Neumann. Estas medidas de información han demostrado ser esenciales para comprender la naturaleza cuántica de la información y sus implicaciones en la computación cuántica, la criptografía y otras disciplinas avanzadas.

La Entropía de Shannon, desarrollada por Claude Shannon en el contexto de la teoría de la información clásica, ha sido una herramienta invaluable para cuantificar la incertidumbre asociada con la información en sistemas clásicos. Por otro lado, la Entropía de von Neumann, concebida por John von Neumann en el ámbito de la mecánica cuántica, se centra en la cuantificación de la incertidumbre en sistemas cuánticos. Ambas entropías comparten la misma base conceptual, pero su aplicación y significado difieren significativamente en el contexto cuántico.

En este trabajo, exploraremos detalladamente las diferencias clave entre la Entropía de Shannon y la Entropía de von Neumann, destacando su relevancia en la información cuántica y cómo estas medidas proporcionan una comprensión más profunda de los fenómenos cuánticos. Además, llevaremos a cabo un ejercicio práctico que ilustrará la aplicación de estas entropías en un escenario cuántico específico, permitiendo así una apreciación más práctica y aplicada de estos conceptos teóricos. Este ejercicio nos brindará la oportunidad de observar cómo estas medidas de información se traducen en el mundo cuántico y cómo impactan en la manipulación y transmisión de información en este peculiar dominio.

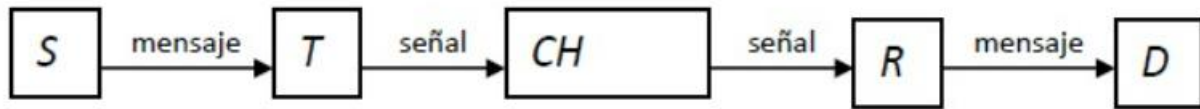
Entropía de Shannon

Fundamentos teóricos

La definición de información fue elaborada por Claude E. Shannon, y presentada en su histórico trabajo “A Mathematical Theory of Communication (1948)” fue con el que sentó las bases de las matemáticas de la teoría de la información y las comunicaciones modernas, este estudio dio el fundamento matemático para industrializar el procesamiento de la información.

De acuerdo a Shannon (1948), un sistema de comunicación general consta de varias partes. Una fuente, la cual genera un mensaje a ser recibido en el destinatario. Un transmisor, que transforma el mensaje generado en la fuente en una señal a ser transmitida. En los casos en los que la información es

codificada, el proceso de codificación también es implementado por el transmisor. Un canal es cualquier medio que sirva para que la señal se transmita desde el transmisor al receptor. Este puede ser, por ejemplo, un cable, una fibra óptica o una señal inalámbrica. Un receptor, que reconstruye el mensaje a partir de la señal, y finalmente, un destinatario, que es quien recibe el mensaje. En el siguiente dibujo se representan estos elementos en forma esquemática:



La fuente S es un sistema que contiene un conjunto de estados diferentes s_1, \dots, s_n , llamados usualmente *letras*. Un aspecto central de la teoría de Shannon es que es posible asignar probabilidades de ocurrencia para los distintos estados de la fuente. Es decir, los estados s_1, \dots, s_n son producidos con probabilidades $p(s_1), \dots, p(s_n)$. Notar que estas distribuciones de probabilidad pueden ser modeladas usando los axiomas de Kolmogorov. La cantidad de información generada por la fuente debido a la ocurrencia del estado s_i se define como:

$$I(s_i) = \log\left(\frac{1}{p(s_i)}\right) = -\log(p(s_i))$$

donde \log es el logaritmo, en base 2 en nuestro caso, que da 0 de sorpresa cuando la probabilidad del suceso es 1. Cuando $P(E)$ es cercana a 1, la sorpresa del suceso es baja, pero si $P(E)$ es cercana a 0, la sorpresa del suceso es alta. El porqué de usar el logaritmo en base 2 es debido a que trabajamos con la unidad de bits. Según sea la base del logaritmo de medida de la información utilizada, esta determinará la unidad de medida de la entropía.

Dado que S produce sucesiones de estados (estas sucesiones son usualmente llamadas mensajes), la entropía de Shannon de la fuente S se define como la cantidad promedio de información producida por la fuente:

$$H(S) = \sum_{i=1}^n p(s_i) \log\left(\frac{1}{p(s_i)}\right) = -\sum_{i=1}^n p(s_i) \log(p(s_i))$$

En forma análoga, el destinatario D es un sistema con un rango de estados posibles d_1, \dots, d_m , a los cuales se le asignan probabilidades $p(d_1), \dots, p(d_m)$.

Vale la pena recalcar que los valores que la variable aleatoria tome, no alteran en nada su grado de incertidumbre. Lo único que determine tal grado de incertidumbre es la cantidad de valores que toma y sus respectivas probabilidades.

En cuanto a la Entropía, la definición de la función entropía, $H(S)$, es el producto de la probabilidad de ocurrencia de un evento por la información asociada a dicho evento $H(S) = I(E) P(E)$. La entropía, $H(S)$, es una magnitud que mide la información por símbolos emitidos desde una fuente. Una rápida interpretación nos permite decir que la entropía de una fuente, es la cantidad de bits por datos emitidos por dicha fuente. Si lo analizamos desde el punto de vista de un observador, que está estudiando dicha fuente, la entropía es la incertidumbre que tiene el mismo respecto del próximo dato que será emitido.

Ejemplo

Vamos a ver un pequeño ejemplo para ver la parte práctica de la teoría. Sea un rompecabezas que tiene 12 bolas idénticas en tamaño y apariencia, pero una tiene un peso extraño (puede ser liviana o pesada). Dispone de un conjunto de balanzas que le darán 3 lecturas posibles:

- Izquierda = Derecha,
- Izquierda > Derecha, o
- Izquierda < derecha

Tenemos que diseñar una estrategia para determinar cuál es la bola extraña y si es más pesada o más ligera. Supongamos que hacemos 6 bolas contra 6 bolas en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 0$ (una de las bolas es pesada o liviana)
- Izquierda > Derecha $\rightarrow 0.5$
- Izquierda < Derecha $\rightarrow 0.5$

Entonces tenemos entropía $H(S) = -(0.5 \cdot \log(0.5) + 0.5 \cdot \log(0.5)) = 1 \text{ bit}$

Supongamos que hacemos 5 bolas contra 5 bolas en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 2/12$ (Cualquiera de las dos bolas en el suelo está defectuosa)
- Izquierda > Derecha $\rightarrow 5/12$ (Simetría)

- Izquierda < Derecha $\rightarrow 5/12$ (Simetría)

Entonces tenemos entropía - $((2/12)*\log(2/12) + (5/12)*\log(5/12) + (5/12)*\log(5/12)) = 1,48 \text{ bits}$

Supongamos que hacemos 4 bolas contra 4 bolas en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 4/12$ (Una de las cuatro bolas en el suelo está defectuosa)
- Izquierda > Derecha $\rightarrow 4/12$ (Simetría)
- Izquierda < Derecha $\rightarrow 4/12$ (Simetría)

Entonces tenemos entropía - $((4/12)*\log(4/12) + (4/12)*\log(4/12) + (4/12)*\log(4/12)) = 1,58 \text{ bits}$

Supongamos que hacemos 3 bolas contra 3 bolas en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 6/12$ (Una de las seis bolas en el suelo está defectuosa)
- Izquierda > Derecha $\rightarrow 3/12$ (Simetría)
- Izquierda < Derecha $\rightarrow 3/12$ (Simetría)

Entonces tenemos entropía - $((3/12)*\log(3/12) + (3/12)*\log(3/12) + (6/12)*\log(6/12)) = 1,5 \text{ bits}$

Supongamos que hacemos 2 bolas contra 2 bolas en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 8/12$ (Una de las ocho bolas en el suelo está defectuosa)
- Izquierda > Derecha $\rightarrow 2/12$ (Simetría)
- Izquierda < Derecha $\rightarrow 2/12$ (Simetría)

Entonces tenemos entropía - $((2/12)*\log(2/12) + (2/12)*\log(2/12) + (8/12)*\log(8/12)) = 1,25 \text{ bits}$

Supongamos que hacemos 1 bola contra 1 bola en la balanza. Entonces tenemos las siguientes probabilidades:

- Izquierda = Derecha $\rightarrow 10/12$ (Una de las diez bolas en el suelo está defectuosa)
- Izquierda > Derecha $\rightarrow 1/12$ (Simetría)
- Izquierda < Derecha $\rightarrow 1/12$ (Simetría)

Entonces tenemos entropía - $((1/12)*\log(1/12) + (1/12)*\log(1/12) + (10/12)*\log(10/12)) = 0,82 \text{ bits}$

Concluimos que 4 vs 4 maximiza el contenido de la información.

Entropía de von Neumann

Fundamentos teóricos

En un sistema cuántico caracterizado por una matriz densidad ρ , la entropía de von Neumann, S , es definida por

$$S = -\text{Tr}[\rho \ln(\rho)]$$

donde el logaritmo natural, \ln , de la matriz densidad es

$$\ln(\rho) = (\rho - 1) - \frac{1}{2}(\rho - 1)^2 + \frac{1}{3}(\rho - 1)^3 + \dots$$

Suponemos que el operador densidad es de un estado puro, así que $\rho = |\Psi\rangle\langle\Psi|$ entonces

$$\begin{aligned}\rho \ln(\rho) &= \rho \left((\rho - 1) - \frac{1}{2}(\rho - 1)^2 + \frac{1}{3}(\rho - 1)^3 + \dots \right) \\ &= \rho(\rho - 1) \left[1 - \frac{1}{2}(\rho - 1) + \frac{1}{3}(\rho - 1)^2 + \dots \right]\end{aligned}$$

Pero como, para un estado puro $\rho^2 = \rho$, el factor en el paréntesis desaparece e implica que

$$S(|\Psi\rangle\langle\Psi|) = 0$$

La entropía de von Neumann para un estado puro es 0, lo que significa que tiene un valor bien definido para alguna propiedad. Considerando una representación donde la matriz densidad es diagonal,

$$\rho = \begin{pmatrix} p_1 & 0 & 0 & \dots \\ 0 & p_2 & 0 & \dots \\ 0 & 0 & p_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Entonces

$$\ln(\rho) = \begin{pmatrix} \ln(p_1) & 0 & 0 & \dots \\ 0 & \ln(p_2) & 0 & \dots \\ 0 & 0 & \ln(p_3) & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

y por lo tanto

$$S = -\text{Tr}[\rho \ln(\rho)] = -\sum_i p_i \ln(p_i)$$

y que es proporcional con la entropía de Shannon. Es interesante que la entropía de von Neumann fue anterior a la entropía de Shannon. Es bastante natural considerar la entropía de von Neumann como una generalización de la entropía de Shannon de la teoría clásica de la información, en la que los estados puros desempeñan el papel de variables aleatorias deterministas.

En vez de hacer un ejemplo de cómo calcular la entropía de von Neumann, veremos un ejercicio práctico.

Enunciado del Problema

Let us consider a two-dimensional space and define the state $|\theta\rangle$ as

$$|\theta\rangle = \cos \frac{\theta}{2} |0\rangle + \sin \frac{\theta}{2} |1\rangle.$$

Let a state matrix ρ be given by

$$\rho = p|0\rangle\langle 0| + (1-p)|\theta\rangle\langle \theta|.$$

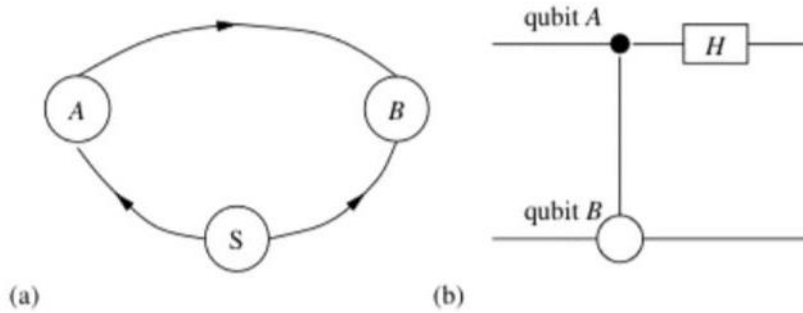


Figure 7.4 (a) General depiction; S, source of entangled particles. (b) Gates applied by Bob.

Compute the Shannon and von Neumann entropies. Show that

$$H_{\text{Sh}} \geq H_{\text{vN}}.$$

Resolución del Problema

La matriz de estado \mathbf{p} se define como:

$$\mathbf{p} = \begin{pmatrix} p + (1-p) \cos^2(\frac{\theta}{2}) & (1-p) \sin(\frac{\theta}{2}) \cos(\frac{\theta}{2}) \\ (1-p) \sin(\frac{\theta}{2}) \cos(\frac{\theta}{2}) & (1-p) \sin^2(\frac{\theta}{2}) \end{pmatrix}$$

Los autovalores de esta matriz (λ_{\pm}) se expresan como:

$$\lambda_{\pm} = \frac{1}{2} (1 \pm \sqrt{1 - 4p(1-p) \sin^2(\frac{\theta}{2})}) = \frac{1}{2} (1 \pm x)$$

La entropía de von Neumann (HvN) se define como:

$$-HvN = (\frac{1+x}{2}) \ln(\frac{1+x}{2}) + (\frac{1-x}{2}) \ln(\frac{1-x}{2})$$

Luego, se calcula la derivada respecto a x de HvN y se obtiene:

$$-\frac{d}{dx} HvN(x) = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right) = \tanh^{-1} x$$

Concluyendo que $HvN(x)$ es una función cóncava de x con un máximo en $x=0$, donde $HvN(x=0)=\ln 2$.

Para este valor de x , se establece que $HvN=Hsh$. Se introduce $p = \frac{1+k}{2}$, y se relaciona Hsh y x mediante la expresión:

$$Hsh = (\frac{1+k}{2}) \ln(\frac{1+k}{2}) + (\frac{1-k}{2}) \ln(\frac{1-k}{2}) \quad \text{y} \quad x = \sqrt{1 - (1-k)^2 \sin^2(\frac{\theta}{2})}$$

Ahora tenemos la desigualdad:

$$k^2 - x^2 = -(1-k^2) \cos^2 \frac{\theta}{2} \leq 0$$

Esto implica que $|x| \geq k$ y $H_{sh} \geq H_{vN}$.

Explicación del Problema

El problema proporciona información sobre la distribución de probabilidad de un sistema cuántico y cómo la entropía cuántica se relaciona con la entropía clásica. Estos resultados pueden tener aplicaciones en el estudio de la información y la teoría cuánticas en general.

La matriz de estado ρ se utiliza en el contexto de la teoría cuántica, donde describe el estado de un sistema cuántico. La variable p representa la probabilidad de que el sistema se encuentre en un cierto estado. La matriz es específica para un parámetro angular (θ) y está relacionada con la rotación de qbits en mecánica cuántica.

Los autovalores de la matriz de estado son soluciones de la ecuación característica, y en este caso, se expresan en función de p y θ . En el problema se calculan los autovalores de la matriz de estado ρ porque estos autovalores son fundamentales para entender y caracterizar las propiedades cuánticas del sistema descrito por dicha matriz.

1. Determinación de Estados Estacionarios:

- En mecánica cuántica, los autovalores de la matriz de estado están relacionados con las energías permitidas del sistema. Al calcular los autovalores, se determinan los estados en los que el sistema puede permanecer indefinidamente en el tiempo.

La entropía de von Neumann es una medida de la cantidad de información (o incertidumbre) asociada con el estado de un sistema cuántico. Aquí hay una explicación de algunos de los elementos de esta fórmula:

- λ : Son los autovalores de la matriz de densidad ρ . Estos autovalores representan las probabilidades de los diferentes estados cuánticos en los que el sistema puede encontrarse. Están relacionados con la probabilidad de encontrar el sistema en un estado particular al realizar una medición.
- \ln : Es la función logaritmo natural. En este contexto, se usa para ponderar las contribuciones de los diferentes autovalores al cálculo de la entropía.

- $\frac{1+x}{2}$ y $\frac{1-x}{2}$: Estos términos representan las probabilidades ponderadas asociadas con los autovalores. La entropía es mayor cuando hay mayor incertidumbre, es decir, cuando las probabilidades asociadas a los autovalores están más equilibradas.

La fórmula busca cuantificar la "mezcla" o "desorden" en el sistema cuántico. Cuando los autovalores son igualmente probables (distribución uniforme), la entropía alcanza su valor máximo, indicando máxima incertidumbre. Por otro lado, cuando un estado cuántico está completamente determinado (un solo autovalor con probabilidad 1), la entropía es cero, indicando certeza total.

Se calcula la derivada de la entropía de von Neumann (HvN) con respecto a x para analizar la concavidad de la función. La razón detrás de este cálculo está relacionada con la naturaleza de la función y la búsqueda de extremos (máximos o mínimos) de HvN en función de x .

La función $-(\frac{d}{dx})(HvN(x))$ se compara con $\tanh^{-1}(x)$. La presencia de la función tangente hiperbólica inversa sugiere que el análisis de extremos está relacionado con funciones hiperbólicas y puede proporcionar información sobre la concavidad de HvN en función de x .

La conclusión inicial es que $HvN(x)$ es una función cóncava de x con un máximo en $x=0$. Esto implica que la gráfica $HvN(x)$ tiene un punto de máximo local cuando $x=0$, y la función es cóncava hacia abajo en ese punto.

Cuando $x=0$, se evalúa $HvN(x=0)$ y se obtiene $\ln 2$. Esto significa que el valor máximo de la entropía de von Neumann ocurre cuando los autovalores están igualmente distribuidos. Además, se establece que para $x=0$, $HvN=Hsh$.

Se introduce $p = \frac{1+k}{2}$ y se relaciona Hsh con x mediante la expresión dada. Esto establece una conexión entre la probabilidad p y los parámetros k y x , mostrando cómo la entropía de Shannon está relacionada con la distribución de probabilidad del sistema cuántico.

Se presenta la desigualdad $k^2 - x^2 = -(1-k^2)\cos^2(\frac{\theta}{2}) \leq 0$. Esta desigualdad implica que $|x| \geq k$, lo que significa que la magnitud de x es mayor o igual a k . Además, se afirma que $Hsh \geq HvN$, lo que indica que la entropía de Shannon es mayor que la entropía de von Neumann en este contexto.

Cuando se afirma que la entropía de Shannon es mayor que la entropía de von Neumann en el contexto del problema que estás analizando, se está haciendo una comparación entre dos medidas de información, una cuántica y otra clásica. Vamos a desglosar lo que esto significa:

La afirmación de que $H_{sh} \geq H_{vN}$ implica que, en el contexto específico del problema, la información promedio (o incertidumbre) en el sistema cuántico, medida por H_{vN} , es menor que la información promedio en el sistema clásico, medida por H_{sh} .

En términos más intuitivos, esta comparación sugiere que la información cuántica contenida en el sistema (cuantificada por H_{vN}) es menos que la información clásica contenida en el mismo sistema (cuantificada por H_{sh}). Esto podría deberse a las peculiaridades cuánticas del sistema, como la superposición cuántica y el entrelazamiento, que pueden hacer que la distribución de probabilidad cuántica sea menos "definida" que su contraparte clásica.

Ampliación del Primer Trabajo

IBM

En IBM Quantum Platform hemos implementado el Algoritmo de Deutsch-Jozsa, uno de los primeros algoritmos diseñados para ejecutar sobre un computador cuántico y que tiene el potencial de ser más eficiente que los algoritmos clásicos al aprovechar el paralelismo inherente de los estados de superposición cuánticos. Lo hemos escogido para probar la plataforma de IBM, usando un oráculo bastante sencillo sólo para probar el funcionamiento.

Aquí hemos importado las clases y funciones necesarias de la biblioteca Qiskit, que hemos ido necesitando a lo largo de la implementación.

```
[2]: from qiskit import QuantumCircuit, Aer, execute
    from qiskit.visualization import plot_histogram
```

Vamos ahora a ver el código principal, comentando línea por línea:

```
circ = QuantumCircuit(4, 3)
```

Crea un circuito cuántico con 4 qubits y 3 bits clásicos para almacenar resultados de mediciones.

```
circ.x(3) # Cambia el estado inicial del último qubit (qubit 3) a |1⟩.
```

Aplica una compuerta X al qubit 3, cambiando su estado inicial de $|0\rangle$ a $|1\rangle$.

```
circ.h([0,1,2,3]) # Aplica compuertas Hadamard a todos los qubits.
```

Aplica compuertas Hadamard a los cuatro qubits. Esto coloca los qubits en una superposición de estados $|0\rangle$ y $|1\rangle$.

```
circ.barrier(range(4))
```

Inserta una barrera en el circuito para visualización. No afecta la ejecución del programa.

```
circ.cx(0,3)
circ.x(0)
circ.cx(0,3)
circ.cx(1,3)
circ.x(1)
circ.cx(1,3)
circ.cx(2,3)
circ.x(2)
circ.cx(2,3)
```

Realiza una serie de operaciones para crear entrelazamiento cuántico entre los qubits 0, 1 y 2 con el qubit 3.

```
circ.barrier(range(4))
```

Inserta otra barrera en el circuito para visualización.

```
circ.h((0,1,2)) # Aplica Hadamard a los qubits 0, 1 y 2.
```

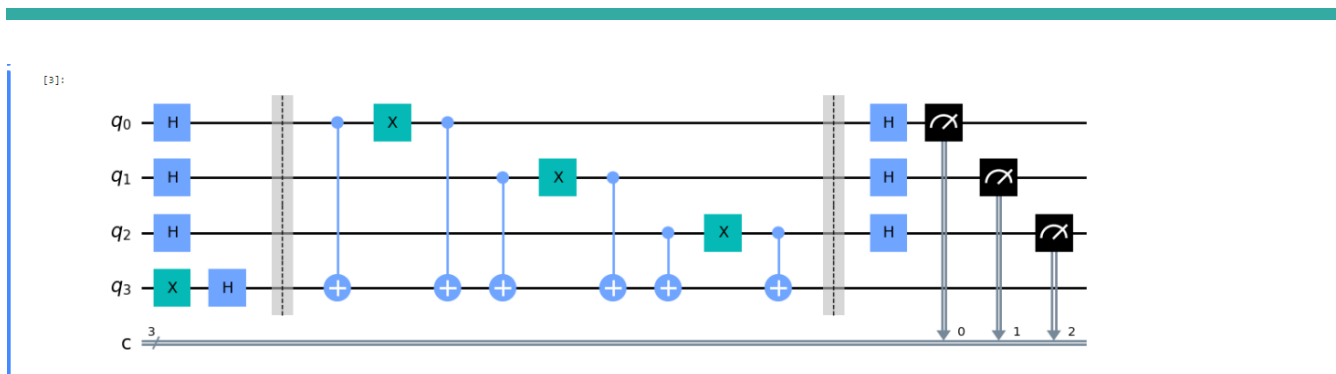
Aplica compuertas Hadamard a los qubits 0, 1 y 2.

```
circ.measure((0,1,2),(0,1,2)) # Realiza una medición de los qubits 0, 1 y 2, y almacena los resultados en los bits clásicos 0, 1 y 2.
```

Realiza mediciones de los qubits 0, 1 y 2 y almacena los resultados en los bits clásicos 0, 1 y 2.

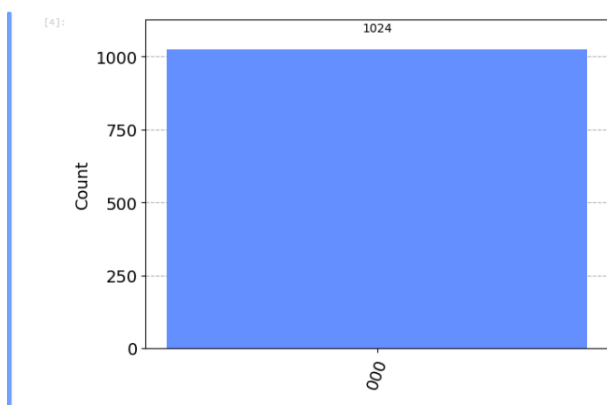
```
[3]: #Número de qubits, y qubits que queremos medir
circ = QuantumCircuit(4, 3)
circ.x(3) # Cambiamos el estado inicial de ese último qubit
circ.h((0,1,2,3)) # Aplicamos Hadamard a todos los qubits
circ.barrier(range(4))
circ.cx(0,3)
circ.x(0)
circ.cx(0,3)
circ.cx(1,3)
circ.x(1)
circ.cx(1,3)
circ.cx(2,3)
circ.x(2)
circ.cx(2,3)
circ.barrier(range(4))
circ.h((0,1,2)) # Aplicamos Hadamard a todos los qubits
circ.measure((0,1,2),(0,1,2)) # medición
circ.draw(output = 'mpl')
```

Para ver de manera más visual lo que estamos haciendo, pues hemos usado la función draw, sobre todo para apoyarnos a la hora de ver resultados. Nos devolvería:



Como sabemos el oráculo que hemos usado es una función constante, por lo tanto es obvio que el resultado que nos dé sea:

```
[4]: backend = Aer.get_backend('qasm_simulator')
job = execute(circ, backend, shots = 1024)
result = job.result()
count = result.get_counts(circ)
plot_histogram(count)
```



Hemos llevado a cabo un proceso iterativo en 1024 ocasiones (llamado "shots"), y en todas las iteraciones el resultado ha sido 000. En otras palabras, como habíamos anticipado previamente, la función es constante.

A parte de trabajar con IBM Quantum Lab, también hemos querido probar el IBM Quantum Composer donde hemos hecho el mismo algoritmo con el mismo oráculo:

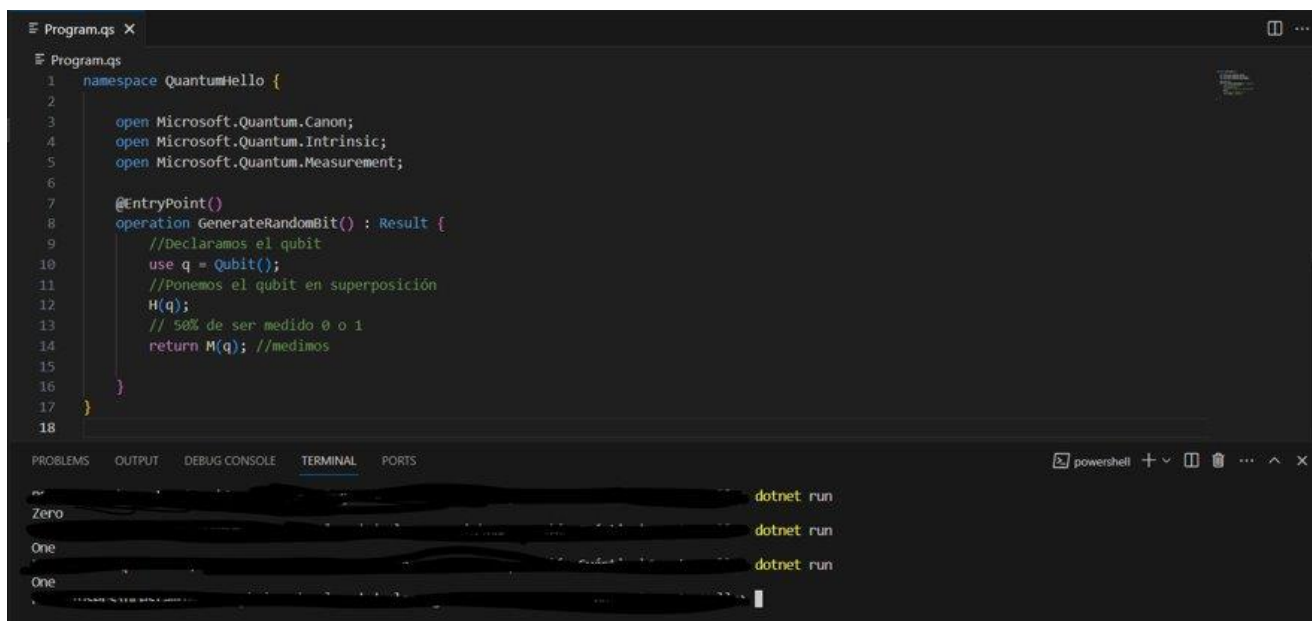


Dándonos exactamente el mismo resultado esperado anteriormente.

Microsoft

En el caso de Microsoft hemos hecho algo más básico, ya que es un lenguaje nuevo Quiskit. En esta plataforma hemos querido aprovechar la superposición de los qubits para generar números realmente aleatorios.

En esta primera función, lo que hacemos es declarar un qubit, ponerlo en superposición aplicándole una puerta Hadamard y medirlo. Al hacer varias iteraciones vemos que nos dará 0 o 1 aleatoriamente con una probabilidad del 50% en ambos casos, si no tenemos en cuenta el ruido que habría.



```
Program.cs
1 namespace QuantumHello {
2
3     open Microsoft.Quantum.Canon;
4     open Microsoft.Quantum.Intrinsic;
5     open Microsoft.Quantum.Measurement;
6
7     @EntryPoint()
8     operation GenerateRandomBit() : Result {
9         //Declaramos el qubit
10        use q = Qubit();
11        //Ponemos el qubit en superposición
12        H(q);
13        // 50% de ser medido 0 o 1
14        return M(q); //medimos
15    }
16 }
17
18
```

Terminal output:

```
dotnet run
Zero
dotnet run
One
dotnet run
One
```

Pero queremos generar números aleatorios desde el 0 hasta un número máximo que nosotros queramos. Lo que hacemos es trabajar en binario, e ir generando bits aleatorios en un rango que contenga el 0 y el número máximo que queramos en binario. Y a partir de ese número en binario que hemos generado, conseguiremos un número completamente aleatorio, que para imprimirlo lo pasaremos a decimal.

```
18
19
20 operation SampleRandomNumberInRange (max : Int) : Int {
21     mutable output = 0;
22     repeat {
23         mutable bits = [];
24         for idxBits in 1..BitSizeI(max) {
25             set bits += [GenerateRandomBit()];
26         }
27         set output = ResultArrayAsInt(bits);
28     }until(output <= max);
29     return output;
30 }
31
32 @EntryPoint()
33 operation SampleRandomNumber() : Int{
34     let max = 50;
35     Message($"Número aleatorio entre 0 y {max}: ");
36     return SampleRandomNumberInRange(max);
37 }
38 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

No se pudo llevar a cabo la compilación. Corrija los errores de compilación y vuelva a ejecutar el proyecto.

Computación Cuántica\QuantumHello> dotnet run

Número aleatorio entre 0 y 50:
35

Computación Cuántica\QuantumHello> dotnet run

Número aleatorio entre 0 y 50:
17

Computación Cuántica\QuantumHello> dotnet run

Número aleatorio entre 0 y 50:
8

Computación Cuántica\QuantumHello> dotnet run

Número aleatorio entre 0 y 50:
9

Computación Cuántica\QuantumHello>

En conclusión, la utilización de la superposición cuántica en Microsoft Azure Quantum nos permite crear secuencias de números verdaderamente aleatorios, destacando la capacidad de los qubits para ofrecer resultados probabilísticos en la generación de números aleatorios controlados.

D-Wave

Nuestro objetivo es mostrar la ayuda del procesador D-Wave con un problema de optimización, para ello hemos elegido el problema del máximo corte. Este problema puede estar presente en situaciones de proyectos del mundo real importantes en el que queremos no depender de un único procesador / servidor y queremos que una parte de ellos se encargue de unas funcionalidades y la otra mitad de otras cosas. Entonces el objetivo es que haya el máximo número de conexiones entre estos conjuntos por si alguno de ellos falla que sea respaldado por otro y no fracase el proyecto.

El problema del máximo corte es un problema de optimización combinatoria que se encuentra en la intersección de la teoría de grafos y la optimización. El objetivo en este problema es encontrar una partición de los nodos de un grafo en dos conjuntos, de modo que la cantidad de aristas que cruzan entre los conjuntos (el corte) sea máxima.

- Primero añadimos las librerías y bibliotecas para implementar el código del problema. Utilizamos también **dwave.system.samplers** para interactuar con un procesador cuántico de D-Wave y **dwave.system.composites** para mapear problemas cuánticos generales a la arquitectura específica de un procesador cuántico de D-Wave. A continuación, nos creamos el grafo y le asignamos las aristas. Automáticamente se añaden los nodos correspondientes al grafo si aún no existen. También inicializamos la matriz Q para el problema QUBO que pueda ser resuelto por el procesador D-Wave.

- Posteriormente, el problema se encarga de ejecutar el problema de optimización cuántica definido por la matriz QUBO (**Q**) en un procesador cuántico de D-Wave. El parámetro **chainstrength** controla la fuerza de acoplamiento entre los qubits y **numruns** especifica el número de ejecuciones independientes que se

realizarán en el procesador cuántico. Cada ejecución puede dar un resultado diferente debido a la naturaleza probabilística de los algoritmos cuánticos. **DWaveSampler()**: Crea una instancia del objeto **DWaveSampler**, que representa el procesador cuántico de D-Wave. **sampler.sample_qubo()**: Esta función envía el problema QUBO al procesador cuántico para su resolución. La variable **response** almacenará los resultados de la ejecución en el procesador cuántico.

```

15 # ----- Import necessary packages -----
16 from collections import defaultdict
17
18 from dwave.system.samplers import DWaveSampler
19 from dwave.system.composites import EmbeddingComposite
20 import networkx as nx
21
22 import matplotlib
23 matplotlib.use("agg")
24 from matplotlib import pyplot as plt
25
26 # ----- Set up our graph -----
27 |
28 # Create empty graph
29 G = nx.Graph()
30
31 # Add edges to the graph (also adds nodes)
32 G.add_edges_from([(1,2),(1,3),(2,4),(3,4),(3,5),(4,5)])
33
34 # ----- Set up our QUBO dictionary -----
35
36 # Initialize our Q matrix
37 Q = defaultdict(int)
38
39 # Update Q matrix for every edge in the graph
40 for i, j in G.edges:
41     Q[(i,i)] += -1
42     Q[(j,j)] += -1
43     Q[(i,j)] += 2
44
45 # ----- Run our QUBO on the QPU -----
46 # Set up QPU parameters
47 chainstrength = 8
48 numruns = 10
49
50 # Run the QUBO on the solver from your config file
51 sampler = EmbeddingComposite(DWaveSampler())
52 response = sampler.sample_qubo(Q,
53                                chain_strength=chainstrength,
54                                num_reads=numruns,
55                                label='Example - Maximum Cut')
56
57 # ----- Print results to user -----
58 print('-' * 60)
59 print('{:>15s}{:>15s}{:>15s}{:>15s}'.format('Set 0','Set 1','Energy','Cut Size'))
60 print('-' * 60)
61 for sample, E in response.data(fields=['sample','energy']):
62     S0 = [k for k,v in sample.items() if v == 0]
63     S1 = [k for k,v in sample.items() if v == 1]
64     print('{:>15s}{:>15s}{:>15s}{:>15s}'.format(str(S0),str(S1),str(E),str(int(-1*E))))
65
66 # ----- Display results to user -----
67 # Grab best result
68 # Note: "best" result is the result with the lowest energy
69 # Note2: the look up table (lut) is a dictionary, where the key is the node index
70 #       and the value is the set label. For example, lut[5] = 1, indicates that
71 #       node 5 is in set 1 (S1).
72 lut = response.first.sample
73
74 # Interpret best result in terms of nodes and edges
75 S0 = [node for node in G.nodes if not lut[node]]
76 S1 = [node for node in G.nodes if lut[node]]
77 cut_edges = [(u, v) for u, v in G.edges if lut[u]!=lut[v]]
78 uncut_edges = [(u, v) for u, v in G.edges if lut[u]==lut[v]]
79
80 # Display best result
81 pos = nx.spring_layout(G)
82 nx.draw_networkx_nodes(G, pos, nodelist=S0, node_color='r')
83 nx.draw_networkx_nodes(G, pos, nodelist=S1, node_color='c')
84 nx.draw_networkx_edges(G, pos, edgelist=cut_edges, style='dashdot', alpha=0.5, width=3)
85 nx.draw_networkx_edges(G, pos, edgelist=uncut_edges, style='solid', width=3)
86 nx.draw_networkx_labels(G, pos)
87
88 filename = "C:/Users/34656/Downloads/maxcut_plot.png"
89 plt.savefig(filename, bbox_inches='tight')
90 print("\nYour plot is saved to {}".format(filename))

```

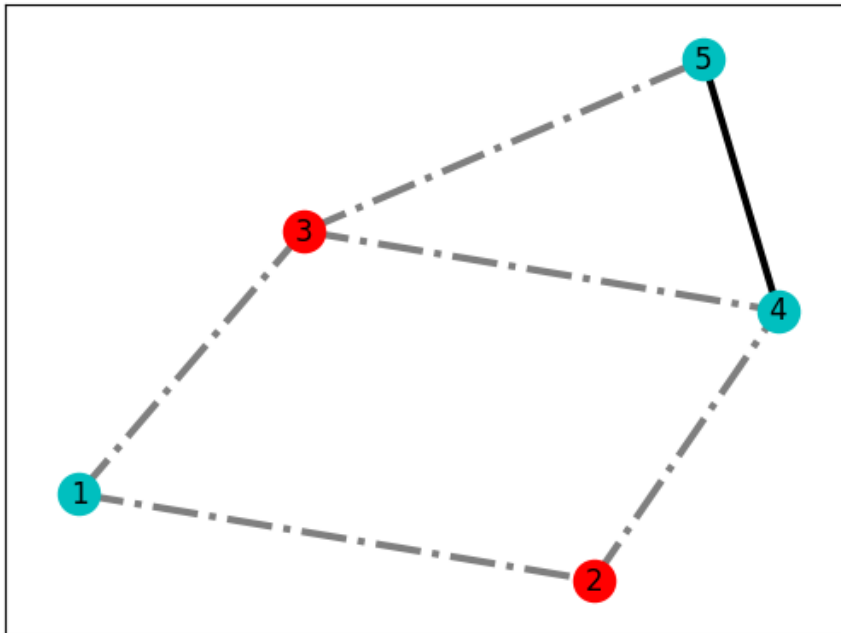
- Luego llevamos a cabo la parte en la que representamos por pantalla los resultados.
- **lut = response.first.sample** obtiene el mejor resultado de la respuesta. En este contexto, "mejor" significa el resultado con la energía más baja asociada. Se interpretan los conjuntos **S0** y **S1** a partir del mejor resultado obtenido. Se identifican las aristas que han sido cortadas (**cut_edges**) y las que no han sido cortadas (**uncut_edges**).
- Por último, la visualización se guarda como un archivo de imagen en el formato PNG en la ubicación especificada por **filename**. Se imprime un mensaje indicando la ubicación donde se ha guardado la visualización.

```
PS C:\Users\34656\Documents\Universidad\topo_comp\new> & C:/Users/34656/AppData\Local\Programs\Python\Python39-64\python.exe C:/Users/34656/AppData\Local\Programs\Python\Python39-64\python.exe new/maximum-cut.py
```

Set 0	Set 1	Energy	Cut Size
[2, 3]	[1, 4, 5]	-5.0	5
[1, 4]	[2, 3, 5]	-5.0	5
[1, 4, 5]	[2, 3]	-5.0	5
[2, 3, 5]	[1, 4]	-5.0	5

```
Your plot is saved to C:/Users/34656/Downloads/maxcut_plot.png
```

Aquí se muestran los mejores resultados a partir del grafo creado anteriormente. Como vemos, aparecen las 4 posibilidades que existen para encontrar 2 subgrafos que tengan el mayor número de conexiones posible. La variable **Energy** se refiere a la energía asociada a una solución particular del problema, y está directamente relacionada con la función objetivo del problema de optimización cuántica. Para poder resolver un problema de maximización es conveniente pasarlo a un problema de minimización, por ello la energía sale negativa pero el resultado es el correcto.



Esta es la imagen PNG que se guarda al ejecutar el programa. Se puede apreciar que las conexiones que hay entre los 2 subconjuntos es 5 y si alguna de ellas fallara habría alguna otra arista para poder conectar ambos subgrafos, lo que hace muy eficaz este problema.

Otro ejemplo:

Variamos las aristas y añadimos la arista (5,6) y (4,6)

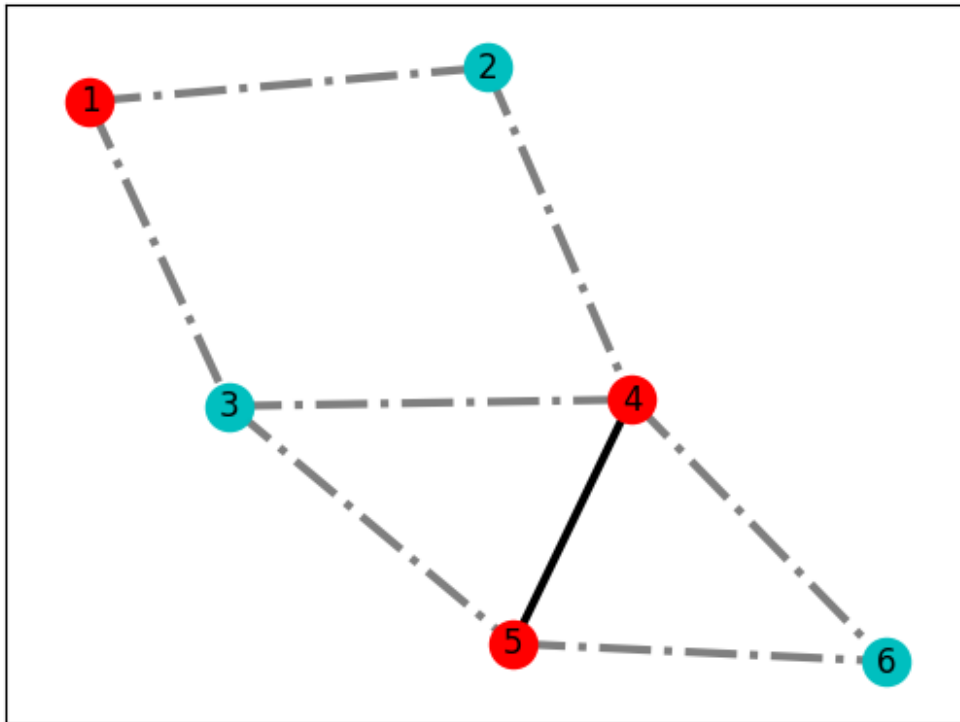
```
31 # Add edges to the graph (also adds nodes)
32 G.add_edges_from([(1,2),(1,3),(2,4),(3,4),(3,5),(4,5),(5,6),(6,4)])
33
```

Nos produce un nuevo resultado:

Set 0	Set 1	Energy	Cut Size
[1, 4, 5]	[2, 3, 6]	-7.0	7
[2, 3, 6]	[1, 4, 5]	-7.0	7

Your plot is saved to C:/Users/34656/Downloads/maxcut_plot.png

Y la foto PNG correspondiente con la resolución visual del problema:



Bibliografía

[1]. Zygelman, B. (2018). *A first introduction to quantum computing and information*.

[2]. Is, W. k. (s/f). 1. *von Neumann Versus Shannon Entropy*. Rickbradford.co.uk.
<http://rickbradford.co.uk/QM6Entropy.pdf>

[3]. Nielsen, M. A., & Chuang, I. L. (2012). *Quantum computation and quantum information*.

[4]. Bellac, M. L. (2006). *A short introduction to quantum information and quantum computation*.