Q10

QX09-DinnerParty.ipynb
File

Necesito que me expliques detalladamente cada una de las celdas del siguiente proyecto

A continuación tienes un desglose detallado, celda por celda, de lo que hace este notebook:

Celda 1: Importaciones y configuración

```
python


from qiskit import *
from qiskit.tools.visualization import plot_histogram
from qiskit.circuit.library import PhaseOracle
from qiskit.algorithms import Grover, AmplificationProblem

%matplotlib inline
```

1. **`from qiskit import *`**: importa las clases y funciones principales de Qiskit (QuantumCircuit, Aer, execute, etc.).
2. **`plot_histogram`**: función de visualización para graficar distribuciones de conteos.
3. **`PhaseOracle`**: crea un oráculo de fase a partir de una expresión booleana; lo usamos para definir qué estados son "buenos".
4. **`Grover` y `AmplificationProblem`**: clases de la nueva API de Qiskit para montar y ejecutar el algoritmo de Grover.
5. **`%matplotlib inline`**: magia de Jupyter para que las gráficas aparezcan embebidas en la misma celda.

Celda 2: Definición del oráculo y ejecución de Grover

```
python


oracle = PhaseOracle('((James & Lars) | (Kirk & Rob)) & ~(Lars & Rob)')
problem = AmplificationProblem(
    oracle=oracle,
    is_good_state=oracle.evaluate_bitstring
)
backend = Aer.get_backend('qasm_simulator')
grover = Grover(quantum_instance=backend)
result = grover.amplify(problem)
```

1. **Expresión booleana**
   - **`((James & Lars) | (Kirk & Rob))`**: selecciona los estados en que o bien (James y Lars) están juntos, o bien (Kirk y Rob) están juntos.

- `& ~(Lars & Rob)`: excluye cualquier estado donde *Lars y Rob* estén simultáneamente.

2. `PhaseOracle(...)` transforma esa fórmula en un circuito que marca (invierte la fase de) los estados "buenos".
3. `AmplificationProblem` encapsula el oráculo y define `is_good_state`, la función que identifica qué bitstrings son soluciones.
4. `Aer.get_backend('qasm_simulator')`: selecciona el simulador de Qiskit para ejecutar circuitos de medición.
5. `Grover(quantum_instance=backend)`: crea el objeto del algoritmo de Grover.
6. `grover.amplify(problem)`: aplica la secuencia de oráculo y difusión de Grover, devolviendo un objeto `result` con los resultados.

Celda 3: Mostrar el primer resultado

```
python

print(result.circuit_results[0])
```

- `result.circuit_results` es una lista de diccionarios de conteos (`{'0000': 3, '1010': 47, ...}`) correspondientes a cada ejecución repetida.
- Al imprimir el índice `[0]`, ves los conteos de la primera ejecución: cuáles bitstrings midió el simulador y cuántas veces (entre todas las repeticiones).

Celda 4: Graficar histograma de los resultados

```
python

plot_histogram(result.circuit_results)
```

- Dibuja un histograma con barras para cada bitstring observado.
- Las barras más altas indican los estados más amplificados (las soluciones al problema), tras las iteraciones de Grover.

Celda 5: Leyenda de correspondencia entre bits y nombres

```
python

# James - Kirk - Lars - Rob
# James - Lars
# James - Kirk - Lars
# Kirk - Rob
# James - Kirk - Rob
```

1. **Primera línea**: orden de los qubits/variables en el bitstring:

```
posición 0 → James
posición 1 → Kirk
```

```
posición 2 → Lars
posición 3 → Rob
```

2. **Líneas siguientes**: interpretan cada combinación "buena" como nombres separados por guiones. Por ejemplo:
   - `# James - Lars` corresponde al bitstring `1010` (James=1, Kirk=0, Lars=1, Rob=0).
   - `# Kirk - Rob` corresponde a `0101`, etc.

Celda 6 a 10: Celdas vacías

- No contienen código ni comentarios. Quedan disponibles para añadir análisis adicional, imprimir estadísticas, o ejecutar más visualizaciones si se desea.

Con esto tienes tanto la lógica de configuración y ejecución de Grover, como la forma de interpretar los resultados en función de los nombres de los participantes. Si quieres profundizar en alguno de estos pasos (por ejemplo, en cómo internamente `PhaseOracle` traduce la fórmula a puertas cuánticas, o en cómo ajustar el número de iteraciones de Grover), dímelo y lo revisamos con más detalle.

Podrias darme mas ejemplos de esto

Aquí tienes tres ejemplos adicionales de cómo formular distintos problemas con Grover y `PhaseOracle`, junto con el código y la explicación de cada paso:

Ejemplo 1: Buscar un único estado marcado en 2 qubits

Problema: de los cuatro posibles estados de 2 qubits (`00, 01, 10, 11`), queremos encontrar el estado `11`.

```python
from qiskit import Aer
from qiskit.tools.visualization import plot_histogram
from qiskit.circuit.library import PhaseOracle
from qiskit.algorithms import Grover, AmplificationProblem

# 1. Definir el oráculo que marca solo el bitstring '11'
oracle = PhaseOracle('(q0 & q1)')

# 2. Crear el problema de amplificación
problem = AmplificationProblem(
    oracle=oracle,
    is_good_state=oracle.evaluate_bitstring
)

# 3. Seleccionar el simulador y el algoritmo
backend = Aer.get_backend('qasm_simulator')
grover = Grover(quantum_instance=backend)

# 4. Ejecutar Grover
result = grover.amplify(problem)

# 5. Visualizar resultados
```

```
print(result.top_measurement)      # debería imprimir '11'
plot_histogram(result.circuit_results)
```

Explicación paso a paso:

1. **Oráculo**: la expresión `(q0 & q1)` evalúa a verdadero únicamente cuando ambos qubits (`q0`, `q1`) están en 1.
2. **AmplificationProblem**: envuelve el oráculo y define qué estados son "buenos".
3. **Grover**: configura el algoritmo para usar el simulador de Qiskit.
4. **Amplify**: ejecuta iteraciones de oráculo + difusión para reforzar la probabilidad de `11`.
5. **Resultado**: al medir, la mayoría de las veces obtendrás `11`.

Ejemplo 2: Buscar todas las combinaciones con exactamente dos 1's en 3 qubits

Problema: en 3 qubits (`000` a `111`), queremos los estados con dos unos: `110`, `101` y `011`.

```
python


from qiskit import Aer
from qiskit.tools.visualization import plot_histogram
from qiskit.circuit.library import PhaseOracle
from qiskit.algorithms import Grover, AmplificationProblem

# Oráculo: satisface exactamente dos qubits a 1
expr = '''
    (q0 & q1 & ~q2)      # 110
  | (q0 & ~q1 & q2)      # 101
  | (~q0 & q1 & q2)      # 011
'''
oracle = PhaseOracle(expr)

problem = AmplificationProblem(
    oracle=oracle,
    is_good_state=oracle.evaluate_bitstring
)

backend = Aer.get_backend('qasm_simulator')
grover = Grover(quantum_instance=backend)
result = grover.amplify(problem)

print("Estados más probables:", result.top_measurement)  # uno de '110','101' o '011'
plot_histogram(result.circuit_results)
```

Puntos clave:

- La expresión booleana es la **disyunción** (`|`) de cada caso donde hay dos qubits a 1.
- Grover amplifica **tres** estados buenos en lugar de uno, por lo que el histograma mostrará picos similares en esas tres combinaciones.

Ejemplo 3: Resolver un mini-3SAT con 3 variables

Problema: hallar asignaciones que satisfacen

$$(a \vee \neg b \vee c) \ \wedge \ (\neg a \vee b \vee \neg c) \ \wedge \ (a \vee b \vee c).$$

```python
from qiskit import Aer
from qiskit.tools.visualization import plot_histogram
from qiskit.circuit.library import PhaseOracle
from qiskit.algorithms import Grover, AmplificationProblem

# Oráculo para la fórmula 3-SAT
sat_expr = '''
    (q0 | ~q1 | q2)   # cláusula 1: a v ¬b v c
  & (~q0 | q1 | ~q2)  # cláusula 2: ¬a v b v ¬c
  & (q0 | q1 | q2)    # cláusula 3: a v b v c
'''
oracle = PhaseOracle(sat_expr)

problem = AmplificationProblem(
    oracle=oracle,
    is_good_state=oracle.evaluate_bitstring
)

backend = Aer.get_backend('qasm_simulator')
grover = Grover(quantum_instance=backend)
result = grover.amplify(problem)

print("Asignación ganadora:", result.top_measurement)
plot_histogram(result.circuit_results)
```

Detalles:

- Cada cláusula se combina con un **AND** (`&`): todas deben ser verdaderas.
- Grover reforzará la(s) asignación(es) de bits que hagan *todas* las cláusulas ciertas.
- El histograma te mostrará cuáles bitstrings (por ejemplo, `101`) son soluciones.

Con estos ejemplos tienes distintos patrones de uso de `PhaseOracle` para:

1. Marcar un único estado.
2. Marcar varios estados que comparten una propiedad (exactamente dos 1’s).
3. Resolver un problema de satisfacción booleana (3-SAT).

Si quieres ver cómo varía el número de iteraciones de Grover, o cómo construir tu propio circuito de difusión, házmelo saber y lo exploramos más a fondo.