Basic Drug Discovery & Compound Activity Prediction exploration using Quantum Computers

Abstract: (Draft Version)

In this report, we will leverage the quantum computers on drug discovery and compound activity prediction life science applications. Quantum computers naturally process the information in sub-atomic level which will plays very important role in healthcare applications.

We will also identify and compare the efficiency and accuracy of quantum sequence alignment using Grover's algorithm with classical sequence alignment methods like smith-Waterman or BLAST using Quantum AI & quantum ML.

Some of the novel idea of this report is to use qubits instead of traditional bits on a sub-atomic level and to understand & to speed up the drug discovery process. And, to predict how certain chemical compound will interact with a biological target, such as a protein or a cell.

Executions Steps:

1. Problem Identification,
2. Collecting Datasets in Classical Computer,
3. Develop the quantum circuits,
4. Execute it in Quantum simulator,
5. Compare the results between quantum computers vs classical super computers.

Python Packages: CIRQ (Google), Qiskit (IBM)

Use Case 3: Model a specific molecule's ground state energy

Objective: Main goal here is to accurately predict the molecule's ground state energy, which could further be used for drug interaction predictions, molecular dynamics, and other applications pivotal in drug discovery.

```
from qiskit_nature.drivers import *
from qiskit_nature.drivers import GaussianDriver

import qiskit_nature.drivers
print(dir(qiskit_nature.drivers))
from qiskit_nature.drivers import PySCFDriver
from qiskit_nature.problems.second_quantization.electronic import ElectronicStructureProblem
from qiskit.algorithms import NumPyMinimumEigensolver
from qiskit_nature.algorithms.ground_state_solvers import GroundStateEigensolver

# Define molecule (Hydrogen molecule)
molecule = 'H .0 .0 .0; H .0 .0 0.735'
driver = PySCFDriver(atom=molecule)

# Define ElectronicStructureProblem
problem = ElectronicStructureProblem(driver)

# Solve with classical eigensolver
solver = NumPyMinimumEigensolver()
calc = GroundStateEigensolver(problem, solver)
result = calc.solve()

# Display result
print("Ground state energy:", result.total_energies[0])
```

Project Information:

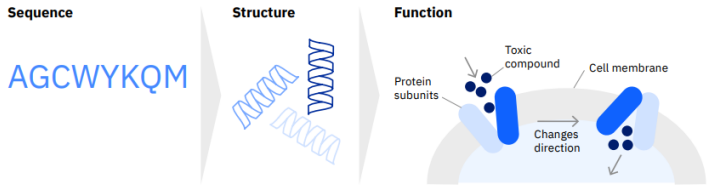
Quantum Computer, working, Principles (Super position, entanglement) circuit building –To be added.

Data collection process for Molecules, Proteins, relevant databases - To be added.

Algorithms & Methods – To be added.

Comparison results – To be added.

(Image Reference: IBM Quantum Life Science Report)



References:

1. https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/quantum-life-sciences
2. https://www.ibm.com/downloads/cas/EVBKAZGJ
3. https://qiskit.org/

Other supporting information: (Rough or supporting usage Only)

Drug discovery:

Drug discovery often involves understanding the interactions between small molecules (potential drugs) and biological targets (like proteins). These interactions happen at the quantum level, and accurately simulating them on classical computers is computationally intensive.

Quantum Applications:

Quantum Molecular Simulations: At the core, drug discovery requires understanding how molecules interact with each other. Quantum computers can simulate these interactions more precisely than classical computers.

Optimization Problems: Finding the best molecule (or drug candidate) out of a vast space of possibilities is an optimization problem. Quantum computers can potentially search this space more efficiently.

Compound Activity Prediction:

Predicting the activity of compounds is essential for identifying potential drug candidates. This involves understanding how a compound might affect biological pathways and predicting its potential therapeutic and side effects.

Quantum Applications:

Machine Learning on Quantum Computers: Quantum-enhanced machine learning can process large datasets of compound activities faster and might detect patterns or correlations that classical algorithms miss.

Complex System Simulations: Biological systems are inherently complex. Quantum computers can simulate intricate systems, like neural networks or metabolic pathways, to predict how a compound might influence them.

Some of the research area's:

Quantum Algorithms for Molecular Docking: Molecular docking is a method used to predict how a molecule (like a drug) will bind to a protein. Designing a quantum algorithm to enhance this process could be revolutionary.

Quantum Machine Learning for Compound Activity Prediction: Implement and compare quantum machine learning models against classical ones using datasets of compound activities. Explore if quantum models offer better accuracy or efficiency.

Hybrid Quantum-Classical Models: Develop models that use quantum computing for specific sub-tasks (like quantum simulations) but rely on classical computing for others (like data preprocessing). Assess the benefits of such a hybrid approach.

Benchmarking and Noise Mitigation: Given the noisy nature of current quantum computers, benchmark quantum algorithms for drug discovery against classical ones. Additionally, explore error mitigation techniques to enhance the reliability of quantum computations.

Exploring Quantum Advantage: Identify specific problems in drug discovery where quantum computers offer a distinct advantage over classical counterparts, either in terms of speed or accuracy.

Use Case 2: Optimization Problem

In layman's terms, think of three friends connected by ropes. The objective is to cut as many ropes as possible by separating the friends into two different groups. Since all three friends are initially tied together, the best you can do is to cut two ropes, separating them effectively into two groups: one group with one friend and another group with the remaining two.

The output of the script tells you the best way to divide these vertices (or friends, in the simpler explanation) to achieve the objective of cutting the most ropes (or edges, in the technical explanation).

Lets execute this in the quantum computer. Simplified Python script using Qiskit that demonstrates the Quantum Approximate Optimization Algorithm (QAOA) for solving a basic optimization problem

```
from qiskit import Aer
from qiskit.algorithms import QAOA
from qiskit.algorithms.optimizers import COBYLA
from qiskit.utils import QuantumInstance
from qiskit_optimization import QuadraticProgram
from qiskit_optimization.algorithms import MinimumEigenOptimizer

# Define the Max-Cut problem as a Quadratic Program
max_cut = QuadraticProgram(name="max_cut")
max_cut.binary_var("x0")
max_cut.binary_var("x1")
max_cut.binary_var("x2")

max_cut.minimize(
    linear=[-1, -1, -1],
    quadratic={
        ("x0", "x1"): 2,
        ("x0", "x2"): 2,
        ("x1", "x2"): 2
    }
)

# Solving the problem classically as a reference
exact_result = MinimumEigenOptimizer(NumPyMinimumEigensolver()).solve(max_cut)
print(f"Exact solution: {exact_result}")

# Set up the quantum instance
backend = Aer.get_backend("aer_simulator")
quantum_instance = QuantumInstance(backend=backend)
```

Use Case 4: Quantum key distribution

Let's dive into quantum key distribution (QKD), a fascinating application that uses quantum properties to secure a communication channel. It's a blend of cryptography and quantum mechanics, very much aligned with real-world scenarios.

Imagine Alice and Bob want to talk without anyone else listening. They use a "secret handshake," but instead of hands, they use special particles called qubits. Alice first prepares these qubits in a way that only she knows. Then she sends them to Bob. When Bob gets them, he tries to figure out Alice's "handshake" to make sure he knows it's really her. If he gets it right, they have a secret way to talk that nobody else can understand.

Here is a simple example using BB84 protocol for QKD:

```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.providers.aer import AerSimulator
from random import randint
import numpy as np

# Initialize variables
n = 10  # Number of qubits, i.e., length of key
alice_bits = [randint(0, 1) for _ in range(n)]

# Create a quantum circuit
qkd_circ = QuantumCircuit(n)

# Alice prepares qubits based on her random bits
for i in range(n):
    if alice_bits[i]:
        qkd_circ.x(i)
    qkd_circ.h(i)

# Bob measures the received qubits
qkd_circ.measure_all()

# Simulate the circuit
simulator = AerSimulator()
compiled_circuit = transpile(qkd_circ, simulator)
job = simulator.run(compiled_circuit)
result = job.result()

# Get measurement statistics
counts = result.get_counts()
```

Use Case 5: Quantum Random Number Generation

Let's explore quantum random number generation. Random numbers are crucial in various applications such as cryptography, simulations, and data analysis. Quantum random number generation can be more "truly random" than classical pseudo-random number generators.

Here's a simple script for quantum random number generation:

```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.providers.aer import AerSimulator
from qiskit.visualization import plot_histogram

# Number of qubits and shots
n = 5
shots = 1024

# Create a quantum circuit with n qubits
circ = QuantumCircuit(n)

# Apply a Hadamard gate on all qubits to create superposition
for i in range(n):
    circ.h(i)

# Measure all qubits
circ.measure_all()

# Simulate the circuit
simulator = AerSimulator()
compiled_circuit = transpile(circ, simulator)
job = simulator.run(compiled_circuit, shots=shots)
result = job.result()

# Get measurement statistics
counts = result.get_counts()

# Visualize the results (optional)
# plot_histogram(counts)

# Extract a random number from the most frequent output
most_frequent = max(counts, key=counts.get)
random_number = int(most_frequent, 2)  # Convert from binary to decimal

print(f"Generated random number: {random_number}")
```

```
Generated random number: 24
```

Why does it important ?. Using a regular computer to make a random number is like having a robot flip a coin for you. Even if it looks