量子计算
—编程篇

Quantum Computer

网址：www.qubits.top

作者：Calvin Tang

邮箱：179209347@qq.com

介绍

本开发教程基于本源量子的Qpanda框架的python版 – **PyQPanda** 编写。

- 一种功能齐全，运行高效的量子软件开发工具包
- QPanda 2是由本源量子开发的开源量子计算框架，它可以用于构建、运行和优化量子算法。
- QPanda 2作为本源量子计算系列软件的基础库，为OriginIR、Qurator、量子计算服务提供核心部件。

QPanda使用文档：
https://pyqpanda-toturial.readthedocs.io/zh/latest/index.html

Github & Gitee 代码地址：
https://github.com/mymagicpower/quantum/tree/main/quantum_qpanda/prog_info
https://gitee.com/mymagicpower/quantum/tree/main/quantum_qpanda/prog_info

NodeIter

NodeIter，是pyQPanda对外提供的 QProg 或者 QCircuit 遍历迭代器，我们可以通过NodeIter很方便的管理我们的量子程序：

获取下一个节点：

```
iter = iter.get_next()
```

获取前项节点：

```
iter = iter.get_pre()
```

获取节点类型：

```
type = iter.get_node_type()
```

通过迭代器构造QProg：

```
type = iter.get_node_type()
if pq.NodeType.PROG_NODE == type:
    prog = pq.QProg(iter)
```

通过迭代器构造量子线路QCircuit：

```
type = iter.get_node_type()
if pq.NodeType.CIRCUIT_NODE == type:
    cir = pq.QCircuit(iter)
```

通过迭代器构造QGate：

```
type = iter.get_node_type()
if pq.NodeType.GATE_NODE == type:
    gate = pq.QGate(iter)
```

通过迭代器构造QIfProg：

```
type = iter.get_node_type()
if pq.NodeType.QIF_START_NODE == type:
    if_prog = pq.QIfProg(iter)
```

通过迭代器构造QWhileProg：

```
type = iter.get_node_type()
if pq.NodeType.WHILE_START_NODE == type:
    while_prog = pq.QWhileProg(iter)
```

通过迭代器构造QMeasure：

```
type = iter.get_node_type()
if pq.NodeType.MEASURE_GATE == type:
    measure_gate = pq.QMeasure(iter)
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/NodeIter.html

NodeIter - 实例

0_NodeIter.py　正向遍历一个QProg

```
import pyqpanda.pyQPanda as pq
import math

machine = pq.init_quantum_machine(pq.QMachineType.CPU)
q = machine.qAlloc_many(8)
c = machine.cAlloc_many(8)
prog = pq.QProg()

prog << pq.H(q[0]) << pq.S(q[2]) << pq.CNOT(q[0], q[1]) \
    << pq.CZ(q[1], q[2]) << pq.CR(q[1], q[2], math.pi/2)
iter = prog.begin()
iter_end = prog.end()
while  iter != iter_end:
   if pq.NodeType.GATE_NODE == iter.get_node_type():
      gate = pq.QGate(iter)
      print(gate.gate_type())
   iter = iter.get_next()
else:
   print('Traversal End.\n')
pq.destroy_quantum_machine(machine)
```

0_NodeIter_reverse.py　反向遍历一个QProg

```
import pyqpanda.pyQPanda as pq
import math

machine = pq.init_quantum_machine(pq.QMachineType.CPU)
q = machine.qAlloc_many(8)
c = machine.cAlloc_many(8)
prog = pq.QProg()

prog << pq.H(q[0]) << pq.S(q[2]) << pq.CNOT(q[0], q[1]) \
       << pq.CZ(q[1], q[2]) << pq.CR(q[1], q[2], math.pi/2)
iter_head = prog.head()
iter = prog.last()
while  iter != iter_head:
   if pq.NodeType.GATE_NODE == iter.get_node_type():
      gate = pq.QGate(iter)
      print(gate.gate_type())
   iter = iter.get_pre()
else:
   print('Traversal End.\n')
```

逻辑门统计

逻辑门的统计是指统计量子程序、量子线路、量子循环控制或量子条件控制中所有的量子逻辑门（这里也会将测量门统计进去）个数方法。

调用接口 get_qgate_num 统计量子逻辑门的个数：

```
number = get_qgate_num(prog)
```

统计 QCircuit 、 QWhileProg 、QIfProg 中量子逻辑门的个数和 QProg 类似。

1_Gate_Counter.py

```
from pyqpanda import *
if __name__ == "__main__":
    qvm = init_quantum_machine(QMachineType.CPU)
    qubits = qvm.qAlloc_many(2)
    cbits = qvm.cAlloc_many(2)
    prog = QProg()
    # 构建量子程序
        prog << X(qubits[0]) << Y(qubits[1])\
             << H(qubits[0]) << RX(qubits[0], 3.14)\
             << Measure(qubits[0], cbits[0])

    # 统计逻辑门个数
     number = get_qgate_num(prog)
    print("QGate number: " + str(number))
    qvm.finalize()
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QGateCounter.html

统计量子程序时钟周期

已知每个量子逻辑门在运行时所需时间的条件下，估算一个量子程序运行所需要的时间。每个量子逻辑门的时间设置在项目的元数据配置文件 QPandaConfig.xml 中， 如果未设置则会给定一个默认值，单量子门的默认时间为2，双量子门的时间为5。

配置文件可仿照下面设置：

```
"QGate": {
    "SingleGate":{
        "U3":{"time":1}
    },
    "DoubleGate":{
        "CNOT":{"time":2},
        "CZ":{"time":2}
    }
}
```

调用 get_qprog_clock_cycle 接口得到量子程序的时钟周期：

```
clock_cycle = get_qprog_clock_cycle(qvm, prog)
```

2_Clock_Cycle.py

```
from pyqpanda import *
import numpy as np

if __name__ == "__main__":
    qvm = init_quantum_machine(QMachineType.CPU)
    qubits = qvm.qAlloc_many(4)
    cbits = qvm.cAlloc_many(4)
    # 构建量子程序
     prog = QProg()
     prog << H(qubits[0]) << CNOT(qubits[0], qubits[1])\
             << iSWAP(qubits[1], qubits[2]) << RX(qubits[3], np.pi / 4)

    # 统计量子程序时钟周期
     clock_cycle = get_qprog_clock_cycle(prog, qvm)
     print(clock_cycle)
     destroy_quantum_machine(qvm)
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QProgClockCycle.html

获取量子线路对应矩阵

接口 get_matrix 可以获得输入线路的对应矩阵，有3个输出参数，一个量子线路QCircuit(或者Qprog)，另外两个是可选参数：迭代器开始位置和结束位置，用于指定一个要获取对应矩阵信息的线路区间，如果这两个参数为空，代表要获取整个量子线路的矩阵信息。
使用 get_matrix 需要注意的是量子线路中不能包含测量操作。

```
import pyqpanda.pyQPanda as pq
import math
class InitQMachine:
    def __init__(self, quBitCnt, cBitCnt, machineType =
pq.QMachineType.CPU):
        self.m_machine = pq.init_quantum_machine(machineType)
        self.m_qlist = self.m_machine.qAlloc_many(quBitCnt)
        self.m_clist = self.m_machine.cAlloc_many(cBitCnt)
        self.m_prog = pq.QProg()
    def __del__(self):
        pq.destroy_quantum_machine(self.m_machine)
```

3_Get_Matrix.py

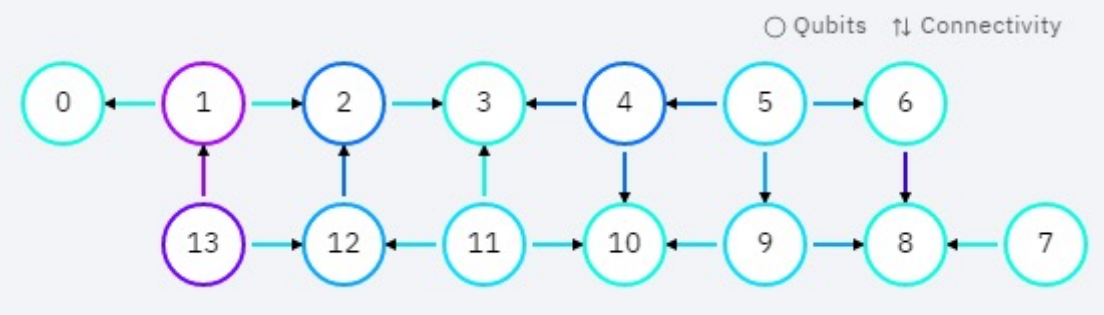
```
def get_matrix(q, c):
    prog = pq.QProg()
    # 构建量子程序
        prog << pq.H(q[0]) \
        << pq.S(q[2]) \
        << pq.CNOT(q[0], q[1]) \
        << pq.CZ(q[1], q[2]) \
        << pq.CR(q[1], q[2], math.pi/2)
    # 获取线路对应矩阵
        result_mat = pq.get_matrix(prog)
    # 打印矩阵信息
        pq.print_matrix(result_mat)

if __name__=="__main__":
    init_machine = InitQMachine(16, 16)
    qlist = init_machine.m_qlist
    clist = init_machine.m_clist
    machine = init_machine.m_machine
    get_matrix(qlist, clist)
    print("Test over.")
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QCircuitInfo.html#id2

Calvin, QQ: 179209347 Mail: 179209347@qq.com

判断量子逻辑门是否匹配量子拓扑结构

每一款量子芯片都有其特殊的量子比特拓扑结构,例如IBMQ 提供的 ibmq_16_melbourne：

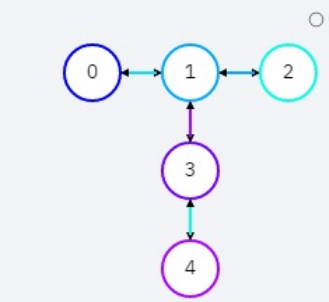


从图中可知，量子芯片中的每个量子比特不是两两相连的，不相连的量子比特之间是不能直接执行多门操作的。所以在执行量子程序之前需要先判断量子程序中的双门（多门）操作是否适配量子比特拓扑结构。

is_match_topology：判断量子逻辑门是否符合量子比特拓扑结构。第一个输入参数是目标量子逻辑门QGate，第二个输入参数是量子比特拓扑结构，返回值为布尔值，表示目标量子逻辑门是否满足量子比特拓扑结构。True为满足，False为不满足。

https://pyqpanda-toturial.readthedocs.io/zh/latest/QCircuitInfo.html#id4

4_Match_Topology.py




```
....
def is_match_topology(q, c):
    cx = pq.CNOT(q[1], q[3])

    # 构建拓扑结构
        qubits_topology =
[[0,1,0,0,0],[1,0,1,1,0],[0,1,0,0,0],[0,1,0,0,1],[0,0,0,1,0]]

    #判断逻辑门是否符合量子拓扑结构
        if (pq.is_match_topology(cx,qubits_topology)) == True:
      print('Match !\n')
    else:
      print('Not match.')

if __name__=="__main__":
    init_machine = InitQMachine(16, 16)
    qlist = init_machine.m_qlist
    clist = init_machine.m_clist
    machine = init_machine.m_machine
    is_match_topology(qlist, clist)
    print("Test over.")
```

获得指定位置的量子逻辑门的相邻量子逻辑门

接口 get_adjacent_qgate_type 可以获得量子程序中指定位置的量子逻辑门的相邻逻辑门。第一个输入参数为目标量子程序QProg，第二个是目标量子逻辑门在量子程序中的迭代器，返回结果是目标量子逻辑门的相邻量子逻辑门迭代器的集合。

get_adjacent_qgate_type 接口的使用方式：

- 构建一个量子程序prog；
- 指定位置信息，即设置iter；
- 调用 get_adjacent_qgate_type 接口获取iter的相邻逻辑门的迭代器集合。示例代码最后4行分别打印了获取到的逻辑门的类型；

5_Adjacent_Qgate_Type.py

```
...
def get_adjacent_qgate_type(qlist, clist):
    prog = pq.QProg()

    # 构建量子程序
        prog << pq.T(qlist[0]) \
                << pq.CNOT(qlist[1], qlist[2]) \
                << pq.Reset(qlist[1]) \
                << pq.H(qlist[3]) \
                << pq.H(qlist[4])

    iter = prog.begin()
    iter = iter.get_next()
    type =iter.get_node_type()
    if pq.NodeType.GATE_NODE == type:
        gate = pq.QGate(iter)
        print(gate.gate_type())
...
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QCircuitInfo.html#id6

判断两个量子逻辑门是否可交换位置

接口 is_swappable 可判断量子程序中两个指定位置的量子逻辑门是否可以交换位置。输入参数一为量子程序 QProg，输入参数二，三是需要判断的两个量子逻辑门的迭代器。返回值为布尔值，True表示可交换，False表示不可交换。

以下实例展示 is_swappable 接口的使用方式：

- 构建一个量子程序prog, 这里列举了一个稍微复杂的带嵌套节点的量子程序；
- 获取嵌套节点cir的两个指定位置的迭代器：iter_first和iter_second；
- 调用 is_swappable 接口判断指定位置的两个逻辑门能否交换位置, 并在控制台输出能否交换的判断结果。

6_Is_Swappable.py

```
...
#测试接口：  判断指定的两个逻辑门是否可以交换位置
def is_swappable(q, c):
    prog = pq.QProg()
    cir = pq.QCircuit()
    cir2 = pq.QCircuit()
    cir2 << pq.H(q[3]) << pq.RX(q[1], math.pi/2) << pq.T(q[2]) <<
    pq.RY(q[3], math.pi/2) << pq.RZ(q[2], math.pi/2)
    cir2.set_dagger(True)
    cir << pq.H(q[1]) << cir2 << pq.CR(q[1], q[2], math.pi/2)
    prog << pq.H(q[0]) << pq.S(q[2]) \
            << cir\
            << pq.CNOT(q[0], q[1]) << pq.CZ(q[1], q[2]) << pq.measure_all(q,c)
    iter_first = cir.begin()
    iter_second = cir2.begin()
    type =iter_first.get_node_type()
    if pq.NodeType.GATE_NODE == type:
        gate = pq.QGate(iter_first)
        print(gate.gate_type())
...
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QCircuitInfo.html#id8

判断逻辑门是否属于量子芯片支持的量子逻辑门集合

量子芯片支持的量子逻辑门集合可在元数据配置文件QPandaConfig.xml中配置。如果我们没有设置配置文件，QPanda会默认设置一个默认量子逻辑门集合。

默认集合如下所示：

```
single_gates.push_back("RX");
single_gates.push_back("RY");
single_gates.push_back("RZ");
single_gates.push_back("X1");
single_gates.push_back("H");
single_gates.push_back("S");
double_gates.push_back("CNOT");
double_gates.push_back("CZ");
double_gates.push_back("ISWAP");
```

我们可以调用接口 is_supported_qgate_type ，判断逻辑门是否属于量子芯片支持的量子逻辑门集合。
is_supported_qgate_type 接口只有一个参数：目标量子逻辑门。

https://pyqpanda-toturial.readthedocs.io/zh/latest/QCircuitInfo.html#id9

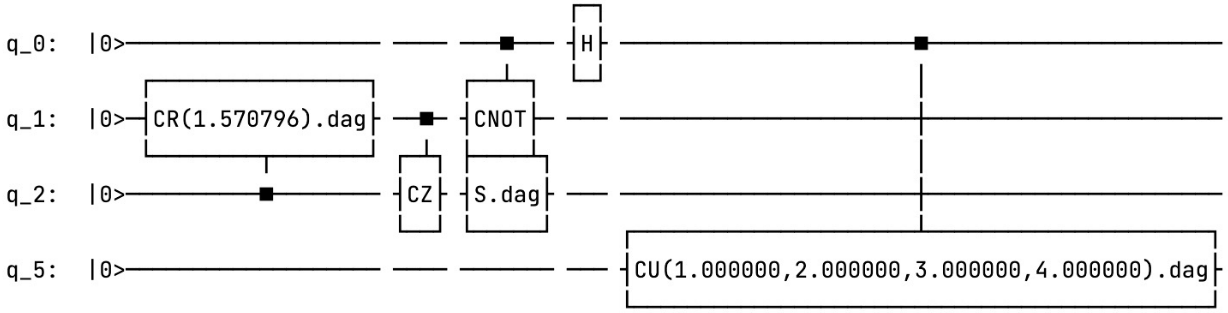
7_Supported_Qgate_Type.py

```
...
def support_qgate_type():
    machine = pq.init_quantum_machine(pq.QMachineType.CPU)
    q = machine.qAlloc_many(8)
    prog = pq.QProg()
    prog << pq.H(q[1])
    result = pq.is_supported_qgate_type(prog.begin())
    if result == True:
        print('Support !\n')
    else:
        print('Unsupport !')

if __name__=="__main__":
    init_machine = InitQMachine(16, 16)
    qlist = init_machine.m_qlist
    clist = init_machine.m_clist
    machine = init_machine.m_machine
    support_qgate_type()
    print("Test over.")
...
```

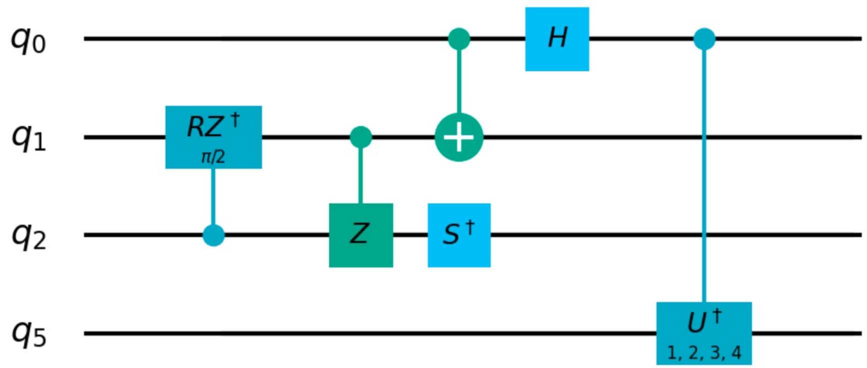
量子线路字符画

目前PyQPanda提供了3中种量子线路可视化方式，具体使用方式参考右侧示例。

示例分别演示了draw_qprog接口的使用方法，代码的输出结果如下：



输出的量子线路图片效果如下：



https://pyqpanda-toturial.readthedocs.io/zh/latest/DrawQProg.html

8_Print_Qcircuit.py

```
...
def print_qcircuit(q, c):
   # 构建量子程序
        prog = pq.QCircuit()
   prog << pq.CU(1, 2, 3, 4, q[0], q[5]) << pq.H(q[0]) << pq.S(q[2])\
          << pq.CNOT(q[0], q[1]) << pq.CZ(q[1], q[2]) << pq.CR(q[2],
          q[1], math.pi/2)
   prog.set_dagger(True)
   print('draw_qprog:')
    print(prog)
   # 通过draw_qprog接口输出量子线路字符画，该方法功能和print方法一样，区别在于该接口可以指定控制台编码类型，以保证在控制台输出的量子线路字符画能正常显示。
   # 参数“console_encode_type”用于指定控制台类型，目前支持两种编码方式:utf8和gbk，默认为utf8
   draw_qprog(prog, 'text', console_encode_type='gbk')
   # draw_qprog接口还可以将量子线路保存成图片，调用方式如下。参数“filename”用于指定保存的文件名。
    draw_qprog(prog, 'pic', filename='./test_cir_draw.png')
...
```

Calvin，QQ: 179209347 Mail: 179209347@qq.com

量子体积

量子体积（**Quantum Volume**），是一个用于评估量子计算系统性能的协议。它表示可以在系统上执行的最大等宽度深度的随机线路。量子计算系统的操作保真度越高，关联性越高，有越大的校准过的门操作集合便会有越高的量子体积。量子体积与系统的整体性能相关联，即与系统的整体错误率，潜在的物理比特关联和门操作并行度相联系。总的来说，量子体积是一个用于近期整体评估量子计算系统的一个实用方法，数值越高，系统整体错误率就越低，性能就越好。

测量量子体积的标准做法就是对系统使用规定的量子线路模型执行随机的线路操作，尽可能地将比特纠缠在一起，然后再将实验得到的结果与模拟的结果进行比较。按要求分析统计结果。

量子体积被定义为指数形式：

$$V_Q = 2^n$$

其中n表示在给定比特数目m（m大于n）和完成计算任务的条件下，系统操作的最大逻辑深度，如果芯片能执行的最大逻辑深度n大于比特数m，那么系统的量子体积就是：

$$2^m$$

量子体积

calculate_quantum_volume 输入参数分别噪声虚拟机或者量子云机器，待测量的量子比特，随机迭代次数，测量次数 。输出为整数，为量子体积大小。

9_Quantum_Volume.py

```
from pyqpanda import *

if __name__=="__main__":
    #构建噪声虚拟机，设置噪声参数
    qvm = NoiseQVM()
    qvm.init_qvm()
    qvm.set_noise_model(NoiseModel.DEPOLARIZING_KRAUS_OPERATOR, GateType.CZ_GATE, 0.005)
    #构建待测量的量子比特组合，  这里比特组合为2组，其中  量子比特3、4为一组；量子比特2，3，5为一组
    qubit_lists = [[3,4], [2,3,5]]
    #设置随机迭代次数
    ntrials = 100
    #设置测量次数,即真实芯片或者噪声虚拟机shots数值
    shots = 2000
    qv_result = calculate_quantum_volume(qvm, qubit_lists, ntrials, shots)
    print("Quantum Volume : ", qv_result)
    qvm.finalize()
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/QuantumVolume.html

Quantum Volume : 8

Calvin, QQ: 179209347 Mail: 179209347@qq.com

随机基准

随机基准测试（RB）是使用随机化方法对量子门进行基准测试。由于完整的过程层析成像对于大型系统是不可行的，因此越来越关注可扩展方法， 以部分表征影响量子系统的噪声。

接口说明:

- single_qubit_rb 的输入参数分别噪声虚拟机或者量子云机器，待测量的量子比特，随机线路clifford门集的不同数量组合、随机线路的数量、测量次数、验证基本逻辑门（默认无），输出为std::map 数据， 关键值为clifford门集数量，数值对应符合期望概率的大小。

- double_qubit_rb 的输入参数分别噪声虚拟机或者量子云机器，待测量的量子比特0，待测量的量子比特1，随机线路clifford门集的不同数量组合、随机线路的数量、测量次数、验证基本逻辑门（默认无）输出为std::map 数据， 关键值为clifford门集数量，数值对应符合期望概率的大小。

10_Qubit_Rb.py

```
from pyqpanda import *
if __name__=="__main__":
    # 构建噪声虚拟机，调整噪声模拟真实芯片
    qvm = NoiseQVM()
    qvm.init_qvm()
qvm.set_noise_model(NoiseModel.DEPOLARIZING_KRAUS_OPERATOR,
GateType.CZ_GATE, 0.005)
qvm.set_noise_model(NoiseModel.DEPOLARIZING_KRAUS_OPERATOR,
GateType.PAULI_Y_GATE, 0.005)
    qv = qvm.qAlloc_many(4)
    # 设置随机线路中clifford门集数量
    range = [ 5,10,15 ]
    # 测量单比特随机基准
    res = single_qubit_rb(qvm, qv[0], range, 10, 1000)
    # 同样可以测量两比特随机基准
    # res = double_qubit_rb(qvm, qv[0], qv[1], range, 10, 1000)
    # 对应的数值随噪声影响，噪声数值越大，所得结果越小，且随
clifford门集数量增多，结果数值越小。
    print(res)
    qvm.finalize()
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/RandomizedBenchmarking.html

Calvin, QQ: 179209347 Mail: 179209347@qq.com

交叉熵基准

交叉熵基准测试（xeb）是一种通过应用随机电路并测量观察到的位串测量值与从模拟获得的这些位串的预期概率之间的交叉熵来评估门性能的方法。

接口说明:

- double_gate_xeb 输入参数分别噪声虚拟机或者量子云机器、待测量的量子比特0、待测量的量子比特1、线路不同层数、随机线路的数量、测量次数、验证双门类型（默认CZ门）。 输出为字典数据，key为线路层数，value对应符合期望概率的大小。

11_Gate_Xeb.py

```
from pyqpanda import *
if __name__=="__main__":
    # 构建噪声虚拟机，调整噪声模拟真实芯片
     qvm = NoiseQVM()
    qvm.init_qvm()
    qv = qvm.qAlloc_many(4)
    # 设置噪声参数
   qvm.set_noise_model(
                        NoiseModel.DEPOLARIZING_KRAUS_OPERATOR,
                        GateType.CZ_GATE, 0.1)
    # 设置不同层数组合
     range = [2,4,6,8,10]
    # 现在可测试双门类型主要为CZ CNOT SWAP ISWAP SQISWAP
    res = double_gate_xeb(qvm, qv[0], qv[1], range, 10, 1000,
                          GateType.CZ_GATE)
    # 对应的数值随噪声影响，噪声数值越大，所得结果越小，且层数
       增多，结果数值越小。

     print(res)
    qvm.finalize()
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/CrossEntropyBenchmarking.html

Calvin, QQ: 179209347 Mail: 179209347@qq.com

Thank You