Reachability Deficits in Quantum Approximate Optimization

项目介绍

命题逻辑的可满足性问题（SAT）是计算机科学中的核心问题。最大可满足问题（Max-SAT）是 SAT 问题的一个自然的扩展。对于给定的 CNF 公式， Max-SAT 问题的目标是找到一个赋值使其满足最多的析取子句。 Max-SAT 是一个重要的NP-难优化问题。由于人工智能、电路自动设计、统计物理、生物信息学等领域的许多问题都可以转化为 Max-SAT 问题，所以近十年来， Max-SAT 问题引起了越来越多的兴趣和关注。

而量子近似优化算法（QAOA）也已迅速成为当代量子计算算法发展的基石。但尽管其应用范围不断扩大，却只有少数研究结果有助于理解该算法的最终局限性。本文发现，当给定近似精度时，与高密度的实例相比，低子句密度的实例所需的 QAOA 电路深度更少。若将临界的 QAOA 深度视为计算成本，电路深度的增加就类似于经典 MAX-SAT 求解器计算资源的需求增加，此时求解器的性能（最佳可能近似值）也会更好。不过，尽管增加深度可以获得更好的近似值，但若固定 QAOA 的深度，其性能（最佳可能近似值）对问题密度 α 的依赖性并不平凡。

本实验以变量数 n=6 的 Max-SAT 为例，对于子句密度 α 从 0 至 10 ，随机各产生 100 组合取范式，调用 QAOA 进行优化，并将结果与实际答案计算误差并作图，很好地说明了目标结论。

复现过程

安装最新版Minduantum

[ ]
```
!pip install https://hiq.huaweicloud.com/download/mindquantum/newest/linux/mindquantum-master-cp37-cp37m-linux_x86_64.whl -i https://pypi.t
```

导入相关依赖

[2]
```
from mindquantum.core import Circuit, Hamiltonian, QubitOperator
from mindquantum.core import UN, H, X, RZ, RX
from mindquantum import ParameterResolver as PR
from mindquantum.framework import MQAnsatzOnlyLayer
from mindquantum.simulator import Simulator
import mindspore as ms
import mindspore.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import random
```

随机生成合取范式

对于合取范式中的 m 条析取子句，首先用 `random.sample` 从 n 个变量中随机选取 k 个（k-SAT问题），再对它们各随机一个 0 或 1 表示对应变量为假或真时不满足条件。当一个析取子句中全部变量均不满足时，意为这个析取子句未被满足，在后文中会产生 1 单位能量作为罚分 penalty 。

[3]
```
def random_clauses(n,m,k=3):
    s = []
    for i in range(m):
        l = random.sample(range(n), k)
        for j in range(k):
            t = random.randint(0, 1)
            l[j] += t * n
        s += [tuple(l)]
    return s
```

经典方法计算实际最小值

由于哈密尔顿量为一个对角矩阵，因此只需统计对角线上最小值，即对于 n 个变量的 2^n 种取值分别计算罚分 penalty，最小值即为最小的未被满足的子句个数。

[4]
```
def calc_minV(n,s,output=False):
    if output:
        print("Perfect answers:")
    ans = len(s)
    for cas in range(1<<n):
        penalty = 0
        for l in s:
            satsfied = False
            for x in l:
                if x < n and cas & (1<<x) != 0 :
                    satsfied = True
                if x >= n and cas & (1<<(x-n)) == 0 :
```

```
                    satsfied = True
            if not satsfied :
                penalty += 1
        if penalty == 0 and output:
            print("  {:0>6}".format(str(bin(cas))[2:]))
        ans = min(ans, penalty)
    return ans
```

搭建量子参数线路

这里我们采用量子绝热近似算法，经过演化将量子态从 $X^{\otimes n}$ 的本征态 $|+\rangle^{\otimes n}$ 演化到合取范式对应哈密尔顿量的基态。

深度为 p 的线路将有 2p 个参数待优化。

[5]
```
def build_ansatz(n,s,p):
    c = UN(H, n)                    # 将|0>生成均匀态
    for i in range(p):
        c += build_Ug(n, s, i)      # 搭建对应哈密顿量的含时演化线路
        c += build_Ub(n, i)         # 搭建 X^n 的含时演化线路
    return c
```

搭建对应哈密顿量的含时演化线路：

[6]
```
def build_Ug(n,s,d,k=3):
    gd = f'g{d}'
    c = Circuit()
    for l in s:
        # 0 ~ n-1  means penalty on 0 (needed flipping)
        # n ~ 2n-1 means penalty on 1
        for x in l:
            if x < n:
                c += X.on(x)
        ll = [x%n for x in l]
        for j in range(k):
            c += RZ(PR({gd: 0.5**j})).on(ll[k-1-j], ll[:k-1-j])
        for x in l:
            if x < n:
                c += X.on(x)
    return c
```

搭建 $X^{\otimes n}$ 的含时演化线路：

[7]
```
def build_Ub(n,d):
    bd = f'b{d}'
    c = Circuit()
    for i in range(n):
        c += RX(bd).on(i)
    return c
```

构建对应的哈密顿量

[8]
```
def build_ham(n,s):
    ham = QubitOperator()
    for l in s:
        mono = QubitOperator('')
        for x in l:
            mono *= (QubitOperator('') + QubitOperator(f'Z{x%n}', (-1)**(x//n))) / 2
        ham += mono
    return ham
```

搭建待训练的神经网络

由于该问题不需要编码层量子线路，我们这里使用MindQuantumAnsatzOnlyLayer作为待训练的量子神经网络，并采用Adam优化器。

以下使用深度 p=15 ，变量数 n=6 ，析取子句数 m=25 时的情况作为演示实例，随机种子依复现要求设置为 42。

[ ]
```
random.seed(42)
p = 15
n = 6
m = 25
s = random_clauses(n, m)
print("s=",s)
minV = calc_minV(n, s, True)
ham = Hamiltonian(build_ham(n, s))
circ = build_ansatz(n, s, p)
ms.context.set_context(mode=ms.context.PYNATIVE_MODE, device_target="CPU")
sim = Simulator('projectq', n)
grad_ops = sim.get_expectation_with_grad(ham, circ)
```

对神经网络进行训练

使得训练次数与线路深度 p 成正比的训练方案：

```
[ ] net = MQAnsatzOnlyLayer(grad_ops)
    opti = nn.Adam(net.trainable_params(), learning_rate=0.05)
    train_net = nn.TrainOneStepCell(net, opti)
    err = train_net().asnumpy()[0]
    for _ in range(p//2):
        err = min(err, train_net().asnumpy()[0])
        print("error =", err-minV)
    err -= minV
    pr = dict(zip(circ.params_name, net.weight.asnumpy()))
    state = circ.get_qs(pr=pr, ket=True)
    print("\nThe Best Parameters:", state, sep='\n')
```

不断训练直至收敛的训练方案：

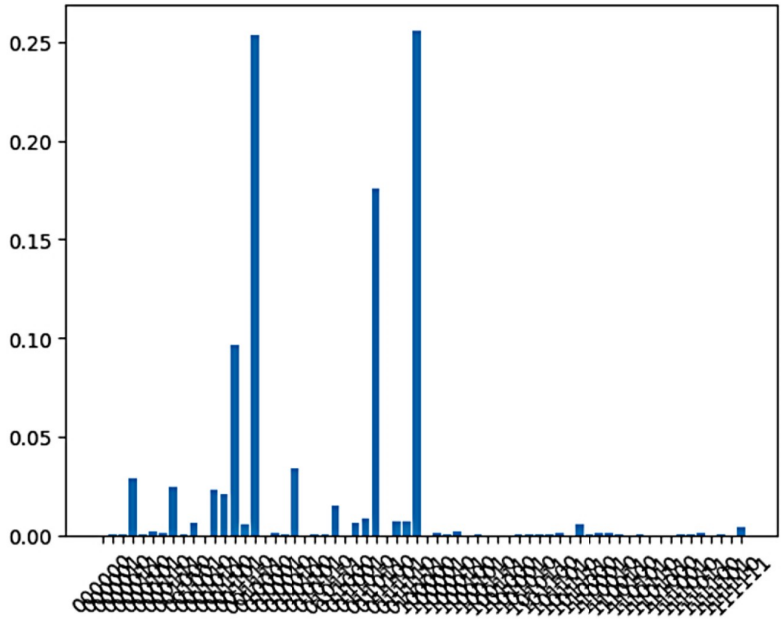
```
[ ] net = MQAnsatzOnlyLayer(grad_ops)
    opti = nn.Adam(net.trainable_params(), learning_rate=0.05)
    train_net = nn.TrainOneStepCell(net, opti)
    lst = train_net().asnumpy()[0]
    while True:
        err = train_net().asnumpy()[0]
        print("error =", err-minV)
        if lst > err and lst - err < 0.001 :
            break
        lst = min(lst, err)
    err -= minV
    pr = dict(zip(circ.params_name, net.weight.asnumpy()))
    state = circ.get_qs(pr=pr, ket=True)
    print("\nThe Best Parameters:", state, sep='\n')
```

对最终结果作图验证

```
[11] import matplotlib.pyplot as plt
    def show_amp(amp):
        n_qubits = int(np.log2(len(amp)))
        labels = [bin(i)[2:].zfill(n_qubits) for i in range(len(amp))]
        plt.bar(labels, amp)
        plt.xticks(rotation=45)
        plt.show()

    state = circ.get_qs(pr=pr)
    amp = np.abs(state)**2
    print(f"\nThe largest amplitude is on |{bin(np.argmax(amp))[2:]:0>6}>\n")
    show_amp(amp)
```

```
The largest amplitude is on |011111>
```



总结

本次复现中，不仅达到了与论文原图较为贴近的效果，还同时设计了多种迭代方案也得到了相近的结论。另外，笔者认为，由于我们的 Max-SAT 问题背景已经决定了**最优结果必为整数**，所以在此先验知识下，调用者完全可以采用 QAOA 优化得到的最优值向下取整作为实际答案。若采用 `max_3_sat_conv.py` 迭代直至收敛的方案，则在取样中 QAOA 优化得到的最优值与实际最优值之差**完全小于 1**，即向下取整完全可以得到正确答案，相比原论文中误差可能大于 1 的情形，其优化效果是从量变到质变的。