本源
量子

QAOA

课程四：算法原理

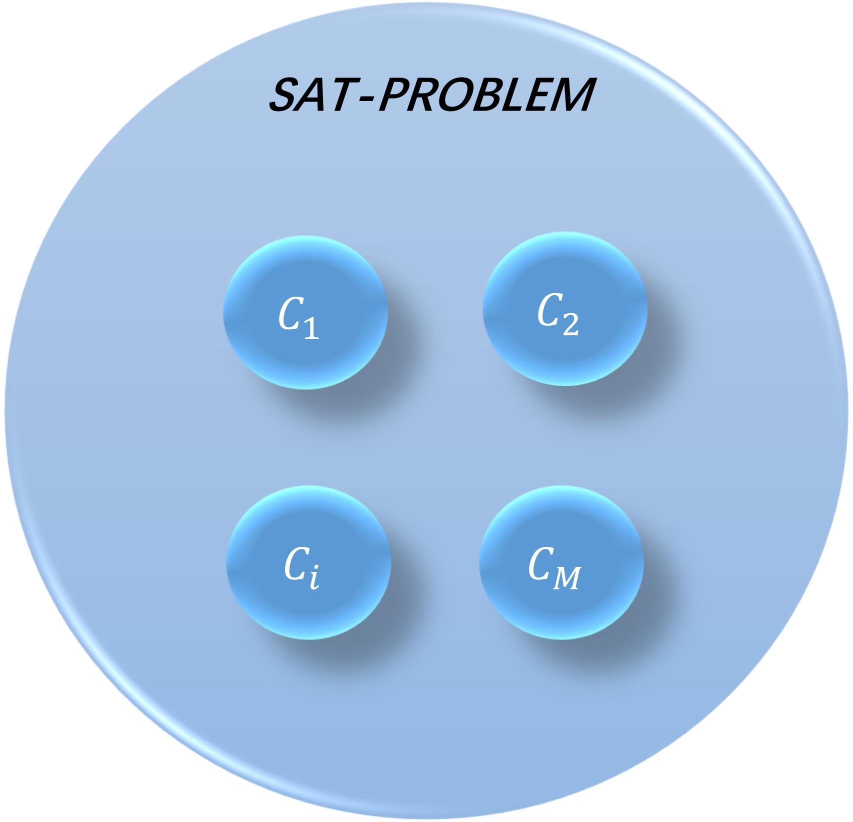
本源量子

QAOA

绝热量子计算

本源量子

QAOA | 绝热量子计算



$$C_1 \wedge C_2 \wedge \cdots \wedge C_M$$

$$C_i = Boolean(x_1, x_2, \dots, x_n) = \begin{cases} true \\ false \end{cases}$$

$(x_i \in \{0,1\})$

$$H_B = H_{B_1} + H_{B_2} + H_{B_i} + \cdots + H_{B_M}$$

$$H_P = \sum_{i=1}^{M} H_{P_i,C_i}$$

$$特征值 h_{C_i} = \begin{cases} 1, & C_i满足条件 \\ 0, & C_i不满足条件 \end{cases}$$

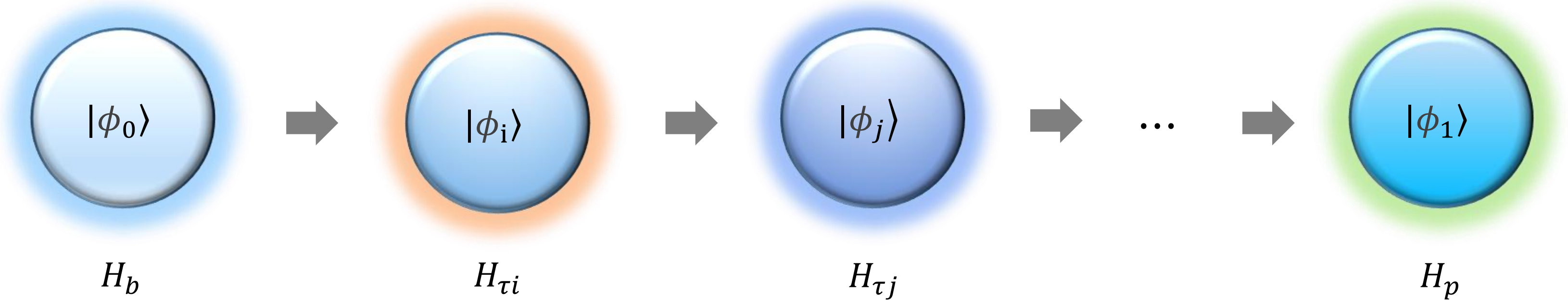
本源量子

QAOA

绝热演化

本源
量子

QAOA | 绝热演化



$$H(t) = \left(1 - \frac{t}{T}\right)H_b + \frac{t}{T}H_p \qquad \text{令}\tau = \frac{t}{T}$$

$$H(\tau) = (1-\tau)H_b + \tau H_p \qquad \tau \in [0,1]$$

$$T = O\left(\frac{\varepsilon}{g_{min}^2}\right), g_{min} = \min_{0\le s\le 1}(E_1(s) - E_0(s))$$

$$\begin{cases} \tau = 0, & \text{静止状态} \\ \tau \neq 0, & \text{演化状态} \end{cases}$$

本源量子

QAOA | 初始哈密顿量

$$H_b = \sum_{i=0}^{n-1} \sigma_i^X$$

$$|\phi_0\rangle = |+\rangle_0 |+\rangle_1 \cdots |+\rangle_{n-1}$$

本源
量子

• QAOA | MaxCut哈密顿量

$$H_p = \sum_{ij} \frac{1}{2}(I - \sigma_i^z \sigma_j^z)$$

本 源
量 子

QAOA | 量子线路

$$U(\vec{\beta},\vec{\gamma}) = \prod_{i=1}^{m} U(H_b,\beta_i)U(H_p,\gamma_i)$$

$$U(H_b,\beta_i) = e^{-iH_b\beta_i}$$

$$U(H_p,\gamma_i) = e^{-iH_p\gamma_i}$$

$$|\phi_1\rangle = |\vec{\beta},\vec{\gamma}\rangle = U(\vec{\beta},\vec{\gamma})\,|\phi_0\rangle$$

本 源
量 子

QAOA | 量子线路

$$U(H_b, \beta_i) = e^{-iH_b\beta_i}$$

$$H_b = \sum_{i=0}^{N-1} \sigma_i^X$$

$$= e^{-i\sum_{n=0}^{N-1}\sigma_n^x \beta_i}$$

$$= \prod_{\mathrm{n}=0}^{N-1} e^{-i\sigma_n^x \beta_i}$$

$$= \prod_{\mathrm{n}=0}^{N-1} RX(n, 2\beta_i)$$

本源
量子

QAOA | 量子线路

$$U(H_p,\gamma_i) = e^{-iH_p\gamma_i}$$

$$H_p = \sum_{ij} \frac{1}{2}(I - \sigma_i^z \sigma_j^z)$$

$$= e^{-i\sum_{jk}\frac{1}{2}(I-\sigma_j^z\otimes\sigma_j^z)\gamma_i}$$

$$= \prod_{jk} e^{-i\frac{\gamma_i}{2}I} \cdot \prod_{jk} e^{i\frac{\gamma_i}{2}\sigma_j^z\otimes\sigma_j^z}$$

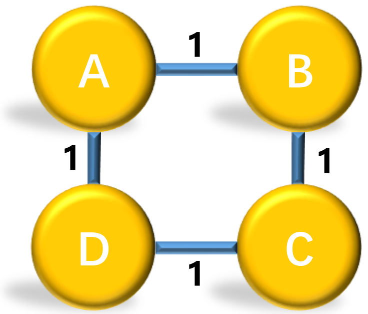
$$e^{i\frac{\gamma_i}{2}\sigma_j^z\otimes\sigma_j^z} = \begin{bmatrix} e^{i\frac{\gamma_i}{2}} & 0 & 0 & 0 \\ 0 & e^{-i\frac{\gamma_i}{2}} & 0 & 0 \\ 0 & 0 & e^{-i\frac{\gamma_i}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\gamma_i}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} e^{i\frac{\gamma_i}{2}} & 0 & 0 & 0 \\ 0 & e^{-i\frac{\gamma_i}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\gamma_i}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\gamma_i}{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

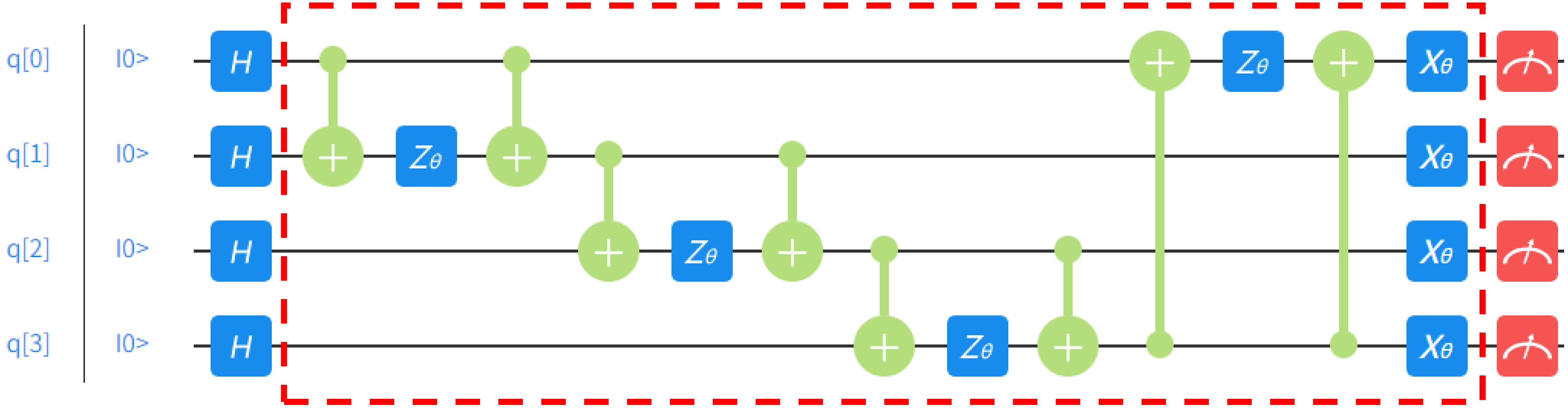
$$\equiv CNOT(j,k)RZ(k,-\gamma_i)CNOT(j,k)$$

QAOA | 量子线路

步数m=1时

$$|\beta,\gamma\rangle = U(\beta_1,\gamma_1)\,|\phi_0\rangle$$
$$= U(H_b,\beta_1)U\big(H_p,\gamma_1\big)\,|\phi_0\rangle$$



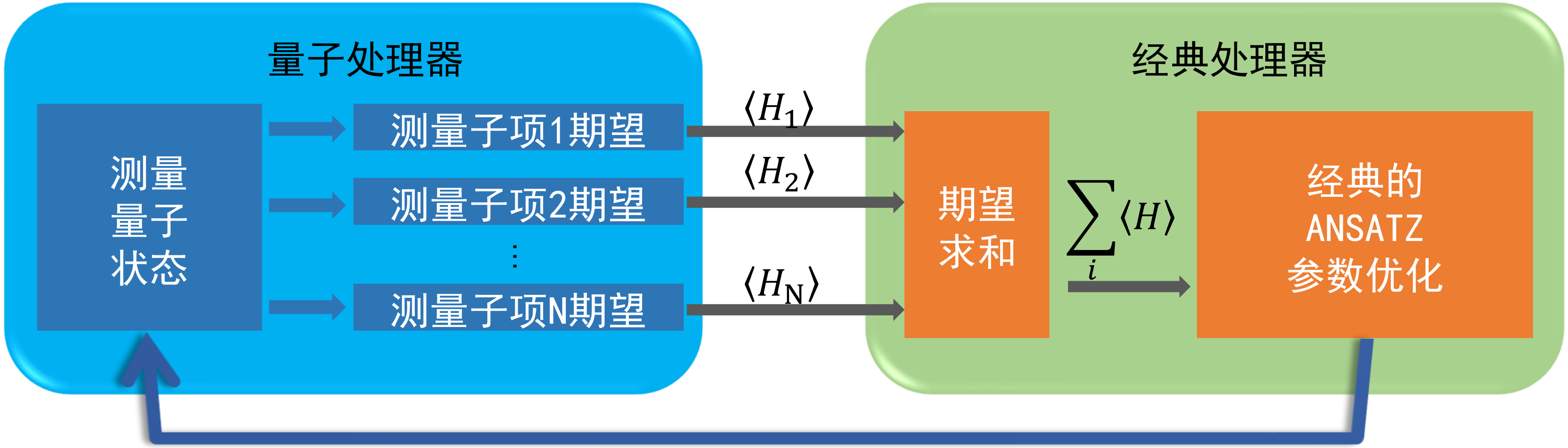


QAOA | 工作流程

本 源
量 子

QAOA | 工作流程

$$Cost(\vec{\beta},\vec{\gamma}) = \langle\phi_0|U^\dagger(\vec{\beta},\vec{\gamma})H_pU(\vec{\beta},\vec{\gamma})|\phi_0\rangle$$



QAOA | 优化算法

- 优化算法
  - 梯度无关
    - Nelder-Mead
    - Powell
    - ⋮
  - 梯度相关
    - Momentum
    - Adam
    - ⋮

本源量子

• QAOA | 梯度无关优化器

```cpp
#include "Optimizer/AbstractOptimizer.h"
#include "Optimizer/OptimizerFactory.h"
int main()
{
   using namespace QPanda;
   auto optimizer = OptimizerFactory::makeOptimizer(NELDER_MEAD);
   vector_d init_para{0, 0};
   optimizer->registerFunc(myFunc, init_para);
   optimizer->setXatol(1e-6);
   optimizer->setFatol(1e-6);
   optimizer->setMaxFCalls(200);
   optimizer->setMaxIter(200);
   optimizer->exec();
   auto result = optimizer->getResult();
   return 0;
}
```

```python
from pyqpanda import *

if __name__=="__main__":

  optimizer =  OptimizerFactory.makeOptimizer(OptimizerType.NELDER_MEAD)
  #optimizer =OptimizerFactory.makeOptimizer('NELDER_MEAD’)

  init_para = [0,0]
  optimizer.registerFunc(myFunc,init_para)
  optimizer.setXatol(1e-6)
  optimizer.setFatol(1e-6)
  optimizer.setMaxFCalls(200)
  optimizer.setMaxIter(200)
  optimizer.exec()
  result = optimizer.getResult()
```

本源量子

QAOA | 梯度相关优化器

```cpp
#include "Variational/Optimizer.h"
int main()
{
    using namespace QPanda::Variational;
    var X(train_x);
    var Y(train_y);
    var W(1.0, true);
    var b(1.0, true);
    var Y_ = W * X + b;
    auto loss = sum(poly(Y - Y_, 2) / train_x.rows());
    auto optimizer = VanillaGradientDescentOptimizer::minimize(loss, 0.01, 1.e-6);
    auto leaves = optimizer->get_variables();
    for (size_t i = 0u; i < 1000; i++)
    {
        optimizer->run(leaves);
        std::cout << "i: " << i << "\t" << optimizer->get_loss()  <<std::endl;
        std::cout << "W:" << QPanda::Variational::eval(W, true)  << std::endl;
        std::cout << "b:" << QPanda::Variational::eval(b, true) << std::endl;
    }
    return 0;
}
```

```python
from pyqpanda import *

if __name__=="__main__":

 X = var(x)
 Y = var(y)
 W = var(1.0, True)
 b = var(1.0, True)
 Y_ = W*X+b
 v = var

 loss = sum(pq.poly(Y - Y_, v(2.0)) / v(17.0))

 optimizer = VanillaGradientDescentOptimizer.minimize(loss, 0.01, 1.e-6)

 leaves = optimizer.get_variables()
 it = 1000

 for i in range(it):
   optimizer.run(leaves)
   oloss = optimizer.get_loss()
   print("i:",i," loss:",oloss," W:",eval(W,True)," b:",eval(b,True))
```

QAOA | 优化器使用演示

追本溯源 高掌远跖

支 持 与 交 流

https://github.com/OriginQ/QPanda-2

https://www.originqc.com.cn