ゲノム解析

量子ゲート方式によるアセンブリング

IBM Community Japan　ナレッジモール研究
量子コンピューターの活用研究 －機械学習・量子化学計算・組み合わせ最適化への応用－

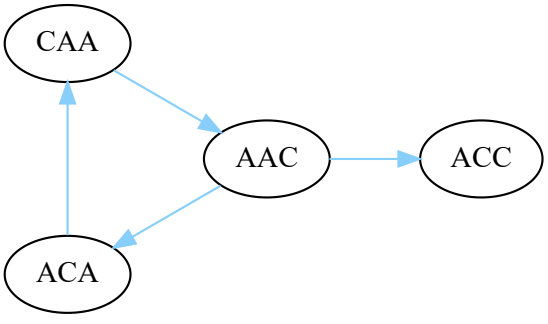
```
# !pip install numpy
# !pip install matplotlib
# !pip install graphviz
# !pip install pymetis
# !pip install networkx
```

```
import numpy as np

from graphviz import Digraph
import networkx as nx
import networkx.algorithms as nxa
import re
import DeBruijnDNA
import matplotlib.pyplot as plt
```

De Bruijn graphのハミルトニアンパスを作成

```
#seq = 'CATACACCTAA' # ゲノムシーケンス
#seq = 'CATACA'
seq = 'ACAACC'
#seq = 'ACATACC'

kmer_len, suffix_len = 3, 2 # 文字数は３、サフィックスは２バイト
# ゲノムシーケンスから隣接グラフとノードラvベルを生成する
adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
g.engine = 'circo'
Q = DeBruijnDNA.to_qubo(adj) # 隣接グラフからquboを生成する
print('solve', len(Q) , 'ising problem')
g # グラフを描画する
```

```
solve 16 ising problem
```




```
''''
```

QAOAで実行する

```
#!pip install qiskit
#!pip install qiskit.optimization
# qiskitの必要なライブラリをimportする
from qiskit import BasicAer
from qiskit.utils import QuantumInstance
```

```
from qiskit.algorithms import QAOA, NumPyMinimumEigensolver
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit_optimization import QuadraticProgram
```

```
# qiskitで必要なqubo式を生成する
qubo = QuadraticProgram()

term_list = ['x_'+ str(i) for i in range(len(Q))]
for term in term_list:
    qubo.binary_var(term)

# numpyのqubo行列から線形項、２次項を取り出して、qiskit用のquboo式を作成する

def get_terms(Q, term_list):
    linear = []
    quadratic = {}
    for i in range(len(Q)):
        for j in range(len(Q)):
            if i==j:
                linear.append(Q[i][j])
            else:
                quadratic[(term_list[i],term_list[j])] = Q[i][j]
    return linear, quadratic

linear, quad = get_terms(Q, term_list)
qubo.minimize(linear=linear, quadratic=quad)
print(qubo.prettyprint()) #qubo式を表示する
```

```
# qubo式をパウリ演算子からなるイジングモデルのハミルトニアンに変換する
op, offset = qubo.to_ising()
print("offset: {}".format(offset))
print("pauli operator:")
print(op) # パウリ演算子からなるハミルトニアンを表示する
```

```
offset: 28.0
pauli operator:
-3.5 * IIIIIIIIIIIIIIIZ
- 4.5 * IIIIIIIIIIIIIIZI
- 4.5 * IIIIIIIIIIIIIZII
- 3.5 * IIIIIIIIIIIIZIII
- 3.5 * IIIIIIIIIIIZIIII
- 4.5 * IIIIIIIIIIZIIIII
- 4.5 * IIIIIIIIIZIIIIII
- 3.5 * IIIIIIIIZIIIIIII
- 3.0 * IIIIIIIZIIIIIIII
- 4.0 * IIIIIIZIIIIIIIII
- 4.0 * IIIIIZIIIIIIIIII
- 3.5 * IIIIZIIIIIIIIIII
- 4.0 * IIIZIIIIIIIIIIII
- 5.0 * IIZIIIIIIIIIIIII
- 5.0 * IZIIIIIIIIIIIIII
- 3.5 * ZIIIIIIIIIIIIIII
+ 0.5 * IIIIIIIIIIIIIIZZ
+ 0.5 * IIIIIIIIIIIIIZIZ
+ 0.5 * IIIIIIIIIIIIIZZI
+ 0.5 * IIIIIIIIIIIIZIIZ
+ 0.5 * IIIIIIIIIIIIZIZI
+ 0.5 * IIIIIIIIIIIIZZII
+ 0.5 * IIIIIIIIIIIZIIIZ
+ 0.5 * IIIIIIIIIIIZIIZI
+ 0.5 * IIIIIIIIIIIZIIIZI
+ 0.5 * IIIIIIIIIIIZIIZII
+ 0.5 * IIIIIIIIIIIZZIIII
+ 0.5 * IIIIIIIIIIZIIIZII
+ 0.5 * IIIIIIIIIIZIIZIII
+ 0.5 * IIIIIIIIIIZIZIIII
+ 0.5 * IIIIIIIIIIZZIIIII
+ 0.5 * IIIIIIIIIZIIIZIII
+ 0.5 * IIIIIIIIIZIIZIIII
+ 0.5 * IIIIIIIIIZIZIIIII
```

```
+ 0.5 * IIIIIIIIZZIIIIII
+ 0.5 * IIIIIIIZIIIIIIIZ
+ 0.5 * IIIIIIIZIIIZIIII
+ 0.5 * IIIIIIIZIIZIIIII
+ 0.5 * IIIIIIZIIIIIIIZ
+ 0.5 * IIIIIIZIIIIIIZI
+ 0.5 * IIIIIIZIIIZIIIII
+ 0.5 * IIIIIIZIIZIIIIII
+ 0.5 * IIIIIIZZIIIIIIII
+ 0.5 * IIIIIZIIIIIIIZI
+ 0.5 * IIIIIZIIIIIIZII
+ 0.5 * IIIIIZIIIZIIIIII
+ 0.5 * IIIIIZIIZIIIIIII
+ 0.5 * IIIIIZIZIIIIIIII
+ 0.5 * IIIIIZZIIIIIIIII
+ 0.5 * IIIIZIIIIIIIZII
+ 0.5 * IIIIZIIIIIIZIII
+ 0.5 * IIIIZIIIZIIIIIII
+ 0.5 * IIIIZIIZIIIIIIII
+ 0.5 * IIIIZIZIIIIIIIII
+ 0.5 * IIIIZZIIIIIIIIII
+ 0.5 * IIIZIIIIIIIZIII
+ 0.5 * IIIZIIIIIIZIIII
+ 0.5 * IIIZIIIZIIIIIIII
+ 0.5 * IIIZIIZIIIIIIIII
+ 0.5 * IIIZIZIIIIIIIIII
+ 0.5 * IIIZZIIIIIIIIIII
+ 0.5 * IIZIIIIIIIIIIIZ
```

```
# イジングモデルのハミルトニアンを２次計画問題に変換
qp = QuadraticProgram()
qp.from_ising(op, offset, linear=True)
print(qp.prettyprint())
```

```
Problem name:

Minimize
  2*x0*x1 + 2*x0*x12 + 2*x0*x13 + 2*x0*x2 + 2*x0*x3 + 2*x0*x4 + 2*x0*x8
  + 2*x0*x9 + 2*x1*x10 + 2*x1*x12 + 2*x1*x13 + 2*x1*x14 + 2*x1*x2 + 2*x1*x3
  + 2*x1*x4 + 2*x1*x5 + 2*x1*x9 + 2*x10*x11 + 2*x10*x13 + 2*x10*x14 + 2*x11*x14
  + 2*x11*x15 + 2*x12*x13 + 2*x12*x14 + 2*x12*x15 + 2*x13*x14 + 2*x13*x15
  + 2*x14*x15 + 2*x2*x10 + 2*x2*x11 + 2*x2*x13 + 2*x2*x14 + 2*x2*x15 + 2*x2*x3
  + 2*x2*x5 + 2*x2*x6 + 2*x3*x11 + 2*x3*x14 + 2*x3*x15 + 2*x3*x6 + 2*x3*x7
  + 2*x4*x12 + 2*x4*x13 + 2*x4*x5 + 2*x4*x6 + 2*x4*x7 + 2*x4*x8 + 2*x5*x12
  + 2*x5*x13 + 2*x5*x14 + 2*x5*x6 + 2*x5*x7 + 2*x5*x8 + 2*x5*x9 + 2*x6*x10
  + 2*x6*x13 + 2*x6*x14 + 2*x6*x15 + 2*x6*x7 + 2*x6*x9 + 2*x7*x10 + 2*x7*x11
  + 2*x7*x14 + 2*x7*x15 + 2*x8*x10 + 2*x8*x11 + 2*x8*x12 + 2*x8*x9 + 2*x9*x10
  + 2*x9*x11 + 2*x9*x12 + 2*x9*x13 - x0 - x1 - x10 - x11 - x12 - x13 - x14 - x15
  - x2 - x3 - x4 - x5 - x6 - x7 - x8 - x9

Subject to
  No constraints

  Binary variables (16)
    x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
```

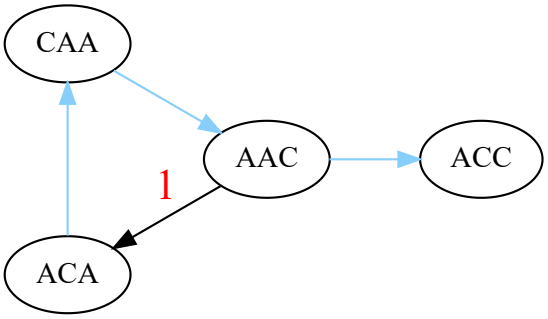
```
# qiskitのローカルシミュレータのインスタンスを生成する
quantum_instance = QuantumInstance(BasicAer.get_backend('qasm_simulator'))
```

```
# qaoaのインスタンスを生成する
qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0.0, 0.0])
qaoa = MinimumEigenOptimizer(qaoa_mes)  # using QAOA
```

```
# QAOAを実行して結果を表示する
%time qaoa_result = qaoa.solve(qubo)
print(qaoa_result.prettyprint())
```

```
CPU times: user 5min 7s, sys: 728 ms, total: 5min 8s
Wall time: 5min 8s
objective function value: -2.0
variable values: x_0=0.0, x_1=1.0, x_2=0.0, x_3=0.0, x_4=0.0, x_5=0.0, x_6=0.0, x_7=0.0, x_8=1.0, x_9=0.0, x_10=0.0, x_
status: SUCCESS
```

```
# 隣接グラフを描画する
g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=qaoa_result.x.tolist(), kmer_len=kmer_len)
g.engine = 'circo'
g
```



''''

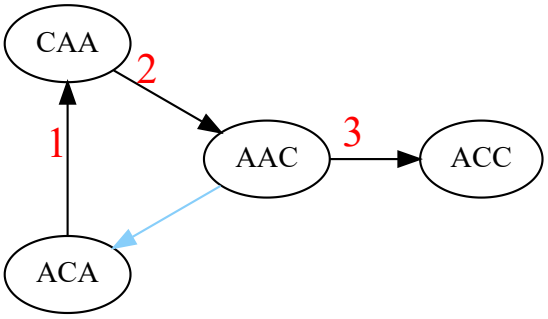
ExactSolverで結果を確認する

```
# ExactSolverのインスタンスを生成する
exact_mes = NumPyMinimumEigensolver()
exact = MinimumEigenOptimizer(exact_mes)  # using the exact classical numpy minimum eigen solver
```

```
# ExactSolverを実行して結果を表示する
exact_result = exact.solve(qubo)
print(exact_result.prettyprint())
```

```
objective function value: -4.0
variable values: x_0=1.0, x_1=0.0, x_2=0.0, x_3=0.0, x_4=0.0, x_5=1.0, x_6=0.0, x_7=0.0, x_8=0.0, x_9=0.0, x_10=1.0, x_
status: SUCCESS
```

```
# 隣接グラフを描画する
g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=exact_result.x.tolist(), kmer_len=kmer_len)
g.engine = 'circo'
g
```



''''