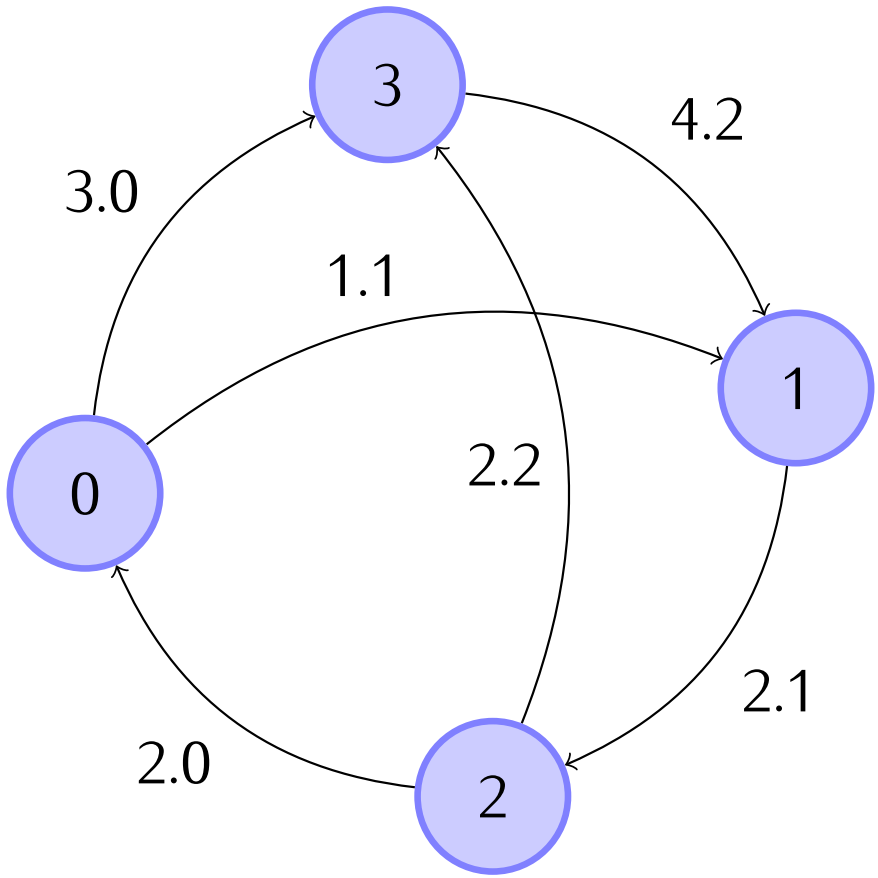
Using XACC-VQE for the 4-City TSP

Stephen Mrenna, FNAL

April 9, 2019

My 4 City World

Paths are bi-directional



4^2 Qubits

	1st	2nd	3rd	4th
n0	0	1	2	3
n1	4	5	6	7
n2	8	9	10	11
n3	12	13	14	15

Several good solutions, all have one $|1\rangle$ per row, one $|1\rangle$ per column
Clockwise and CCW are both good solutions

4^2 Qubits

	1st	2nd	3rd	4th
n0	0	1	2	3
n1	4	5	6	7
n2	8	9	10	11
n3	12	13	14	15

Several good solutions, all have one $|1\rangle$ per row, one $|1\rangle$ per column
Clockwise and CCW are both good solutions

4^2 Qubits

	1st	2nd	3rd	4th
n0	0	1	2	3
n1	4	5	6	7
n2	8	9	10	11
n3	12	13	14	15

Several good solutions, all have one $|1\rangle$ per row, one $|1\rangle$ per column
Clockwise and CCW are both good solutions

4^2 Qubits

	1st	2nd	3rd	4th
n0	0	1	2	3
n1	4	5	6	7
n2	8	9	10	11
n3	12	13	14	15

Several good solutions, all have one $|1\rangle$ per row, one $|1\rangle$ per column
Clockwise and CCW are both good solutions

In [1]:
```
import sys
sys.path.append('/root/.xacc')
import xacc, xaccvqe as vqe
from xaccvqe import PauliOperator
```

In [2]:
```
n = 4  !4 cities
n2 = n*n  !16 qubits
A = PauliOperator(4.2)  !largest dij
term1 = PauliOperator()  !1 = X0 + X1 + X2 + X3
for i in range(0,n2,n):
    temp0 = PauliOperator(1)
    for j in range(i,i+n,1):
        tmp = {}
        tmp[j] = 'Z'
        temp0 = temp0 - (PauliOperator(0.5)-PauliOperator(tmp,0.5))
    term1 = term1 + temp0*temp0
term2 = PauliOperator()  !1 = X0 + X5 + X10 + X15
for i in range(0,n,1):
    temp0 = PauliOperator(1)
    for j in range(0,n2,n):
        tmp = {}
        tmp[i+j] = 'Z'
        temp0 = temp0 - (PauliOperator(0.5)-PauliOperator(tmp,0.5))
    term2 = term2 + temp0*temp0
constraints = A*(term1 + term2)
```

```
(0.5,0) Z3 Z10 + (0.525,0) Z4 Z11 + (0.5,0) Z0 Z11 + (0.75,0) Z1 Z12
+ (0.5,0) Z1 Z10 + (0.5,0) Z0 Z9 + (0.5,0) Z1 Z8 + (0.55,0) Z9 Z12
+ (1.05,0) Z5 Z14 + (0.275,0) Z0 Z5 + (0.525,0) Z7 Z10 + (0.275,0) Z2 Z5
+ (0.55,0) Z10 Z13 + (0.75,0) Z2 Z13 + (0.55,0) Z9 Z14 + (0.75,0) Z2 Z15
+ (1.05,0) Z7 Z12 + (0.75,0) Z0 Z13 + (0.75,0) Z0 Z15 + (0.525,0) Z4 Z9
+ (0.525,0) Z5 Z8 + (0.5,0) Z2 Z11 + (2.1,0) Z1 Z3 + (2.1,0) Z1 Z2
+ (0.525,0) Z5 Z10 + (2.1,0) Z9 Z13 + (0.275,0) Z3 Z4 + (96.4,0)
+ (0.5,0) Z3 Z8 + (2.1,0) Z5 Z6 + (0.5,0) Z2 Z9 + (2.1,0) Z5 Z13
+ (-11.45,0) Z2 + (0.275,0) Z3 Z6 + (2.1,0) Z9 Z10 + (2.1,0) Z3 Z7
+ (2.1,0) Z13 Z15 + (2.1,0) Z13 Z14 + (2.1,0) Z1 Z13 + (2.1,0) Z10 Z14
+ (2.1,0) Z3 Z11 + (2.1,0) Z8 Z10 + (2.1,0) Z6 Z14 + (1.05,0) Z4 Z15
+ (0.55,0) Z11 Z12 + (-11.55,0) Z8 + (2.1,0) Z11 Z15 + (2.1,0) Z2 Z14
+ (2.1,0) Z5 Z9 + (2.1,0) Z0 Z8 + (2.1,0) Z7 Z11 + (2.1,0) Z0 Z12
+ (-12.1,0) Z5 + (2.1,0) Z12 Z15 + (2.1,0) Z6 Z10 + (2.1,0) Z4 Z12
+ (2.1,0) Z3 Z15 + (0.275,0) Z2 Z7 + (2.1,0) Z1 Z5 + (0.525,0) Z6 Z9
+ (2.1,0) Z7 Z15 + (-12.1,0) Z7 + (2.1,0) Z0 Z2 + (0.55,0) Z8 Z13
+ (-11.45,0) Z0 + (-13.1,0) Z13 + (2.1,0) Z2 Z3 + (2.1,0) Z0 Z1
+ (1.05,0) Z5 Z12 + (2.1,0) Z8 Z12 + (2.1,0) Z14 Z15 + (2.1,0) Z2 Z6
+ (-11.45,0) Z3 + (2.1,0) Z6 Z7 + (2.1,0) Z4 Z7 + (2.1,0) Z12 Z14
+ (0.275,0) Z1 Z6 + (2.1,0) Z0 Z4 + (2.1,0) Z4 Z5 + (2.1,0) Z1 Z9
+ (-11.55,0) Z10 + (2.1,0) Z2 Z10 + (-12.1,0) Z4 + (0.275,0) Z0 Z7
+ (2.1,0) Z8 Z9 + (2.1,0) Z8 Z11 + (2.1,0) Z0 Z3 + (0.75,0) Z3 Z12
+ (-11.55,0) Z11 + (-12.1,0) Z6 + (2.1,0) Z9 Z11 + (2.1,0) Z4 Z6
+ (-11.55,0) Z9 + (2.1,0) Z12 Z13 + (2.1,0) Z10 Z11 + (0.75,0) Z3 Z14
+ (-13.1,0) Z14 + (-13.1,0) Z12 + (0.55,0) Z8 Z15 + (0.55,0) Z11 Z14
+ (2.1,0) Z4 Z8 + (-11.45,0) Z1 + (1.05,0) Z6 Z15 + (0.75,0) Z1 Z14
+ (2.1,0) Z5 Z7 + (1.05,0) Z6 Z13 + (1.05,0) Z4 Z13 + (0.275,0) Z1 Z4
+ (0.525,0) Z7 Z8 + (-13.1,0) Z15 + (0.55,0) Z10 Z15 + (1.05,0) Z7 Z14 + (0.525,0) Z6 Z11
```

Ansatz: one y-rotation per qubit

Switches from ↓ to ↑

```
In [5]: xacc.Initialize()
        xacc.setOption("itensor-svd-cutoff", 1e-4)
        tnqvm = xacc.getAccelerator('tnqvm')
        buffer = tnqvm.createBuffer('q', 16)
```

```
In [40]: @xacc.qpu(algo='vqe',accelerator=tnqvm,observable=ham4,
optimizer='scipy-COBYLA', options={'disp': True, 'maxiter': 10000, 'tol': 1e-3})
def myhwe(buffer, *args):
     Ry(t0,0)
     Ry(t1,1)
     Ry(t2,2)
     Ry(t3,3)
     Ry(t4,4)
     Ry(t5,5)
     Ry(t6,6)
     Ry(t7,7)
     Ry(t8,8)
     Ry(t9,9)
     Ry(t10,10)
     Ry(t11,11)
     Ry(t12,12)
     Ry(t13,13)
     Ry(t14,14)
     Ry(t15,15)
```

Cartesian Rotation Gates

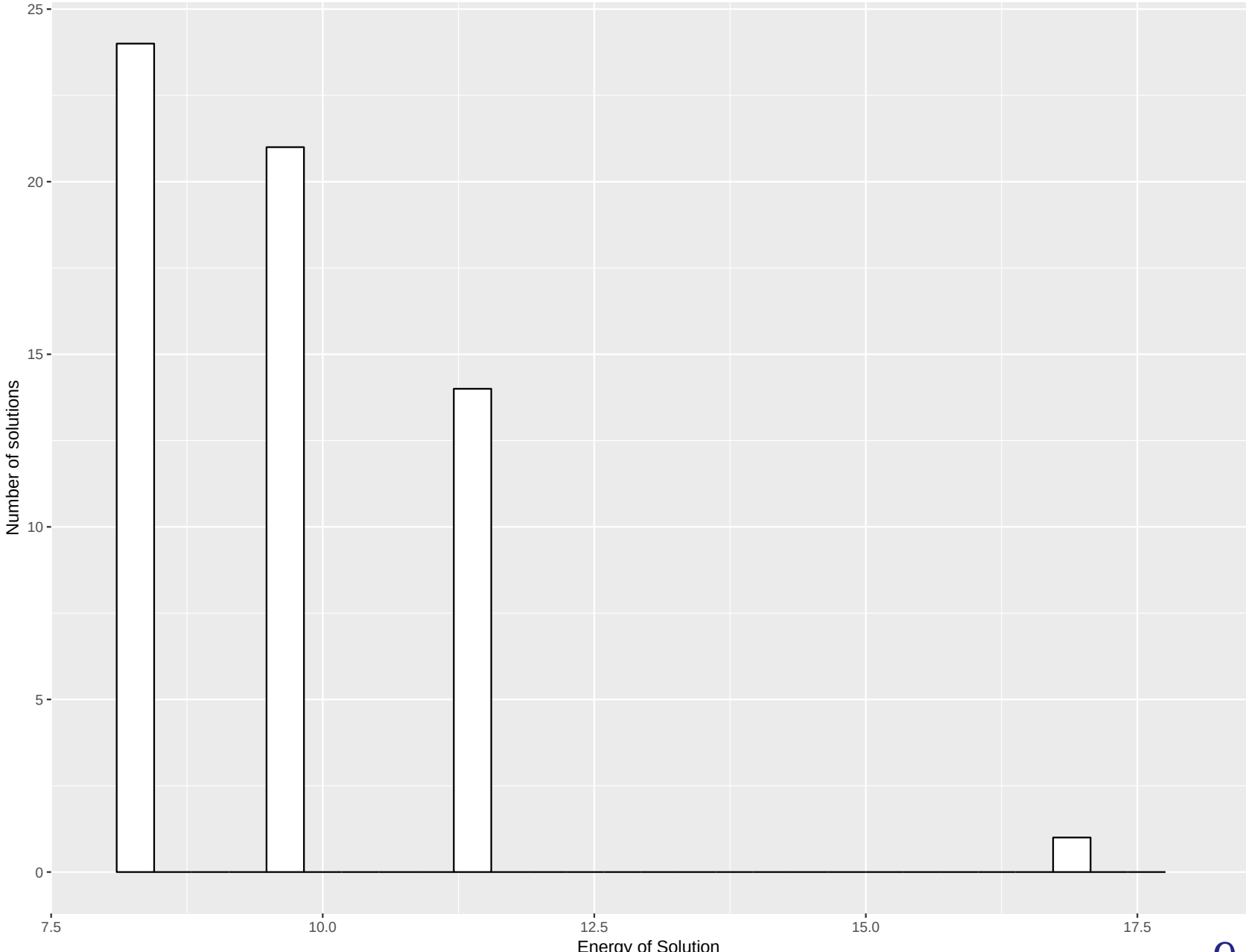
$$\mathtt{RX}(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\mathtt{RY}(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\mathtt{RZ}(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

```
In [41]: init = np.random.uniform(low=-np.pi,high=np.pi, size=(myhwe.nParameters(),))

16
[ 0.84292564 -0.28333584 -1.21276862  2.11376697  0.04167897  0.66223927
  1.580863    1.36225417  2.63549167 -2.38726064  0.97431346  2.10080703
  0.00916234 -0.06945956  1.68138094 -0.71366169]


In [ ]: t = myhwe(buffer, init)

[2019-03-21 15:50:42.429] [xacc-logger] [info] Iteration 4391,
Computed VQE Energy = 11.30000869 at
( -0.000849603003   0.0004072062766  0.0006212523804      3.142400849
  -0.0002237085112 -3.141053187     -8.979674107e-05     -1.718068223e-05
   3.142646014      0.000205111064  -3.718791749e-05      0.0003767434983
  -0.0001547533069  0.0003543149969  3.144496803         -8.733663538e-05)

   Normal return from subroutine COBYLA

   NFVALS =  484   F = 1.130001E+01    MAXCV = 0.000000E+00
   X =-8.496030E-04   4.072063E-04   6.212524E-04   3.142401E+00  -2.237085E-04
      -3.141053E+00  -8.979674E-05  -1.718068E-05   3.142646E+00   2.051111E-04
      -3.718792E-05   3.767435E-04  -1.547533E-04   3.543150E-04   3.144497E+00
      -8.733664E-05
```

Distribution of Solutions



→ Always get a good solution, but not necessarily the best

→ Is there a more efficient parametrization that respects relations like $1 = X0 + X1 + X2 + X3$?

→ Is this minimal QAOA with 2 angles $(e^{iH_x\theta_x} e^{iH_i\theta_i})$?