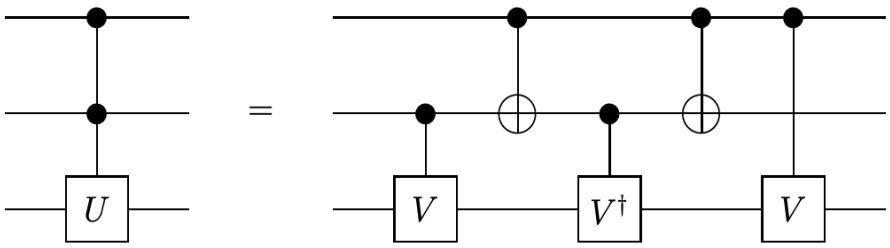
Approximate Toffoli

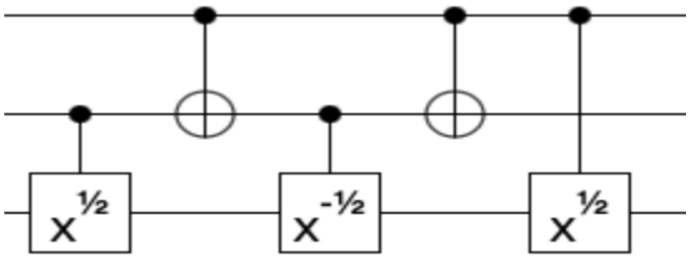
Rishab Khurana

March 2024

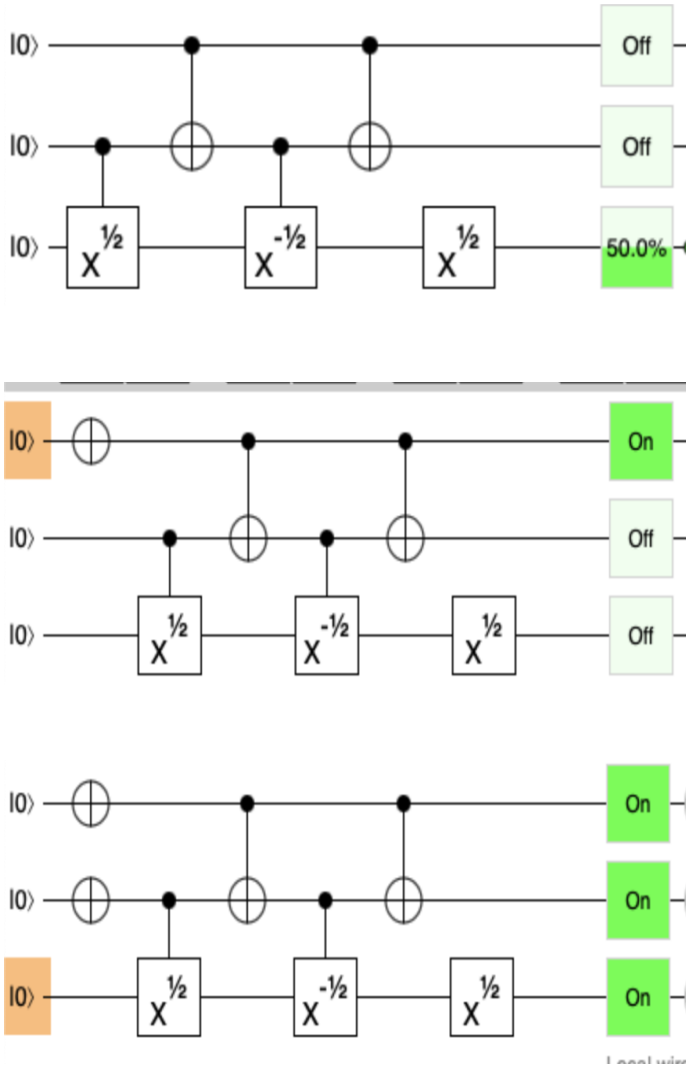
Our goal is to come up with a Toffoli approximation that has a Hilbert-Schmidt distance of about 0.382604. We first start with a decomposition of Toffoli. Barenco et. al. discusses in his paper (link: Elementary gates for quantum computation) the Sleator-Weinfurter Construction:



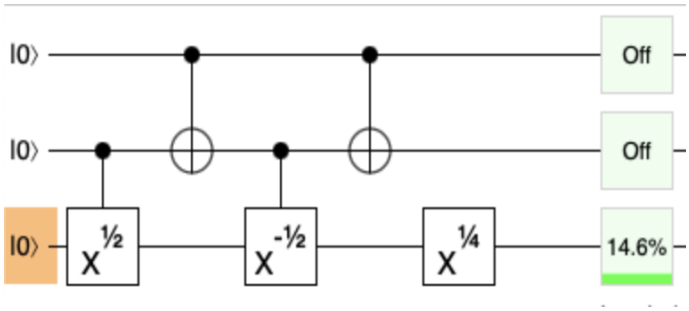
where $V^2 = U$. In the case of this assignment, we have that $U = X$, meaning $V = X^{1/2}$. We thus have the following circuit to implement toffoli *exactly*:

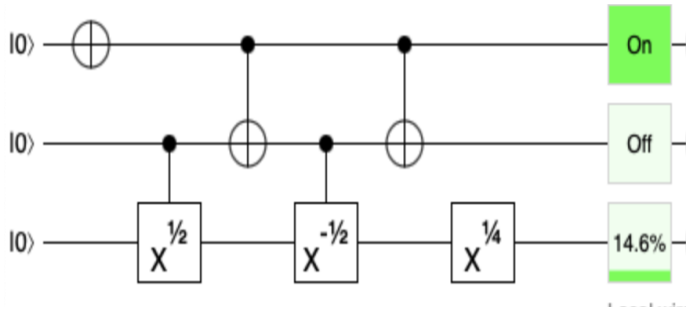


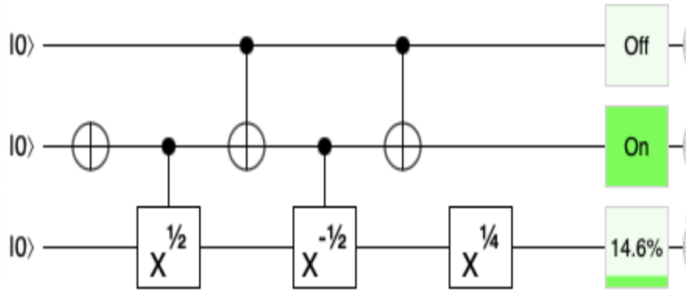
The restriction that the above circuit breaks is the connectedness of the qubits. The first qubit cannot have a two-qubit control with the last qubit being the target bit because our system does not have full connectedness. Thus, we must replace the last gate with gates that give us a sufficient approximation to Tofolli. If we remove the control on the last gate, then we have the following behavior:

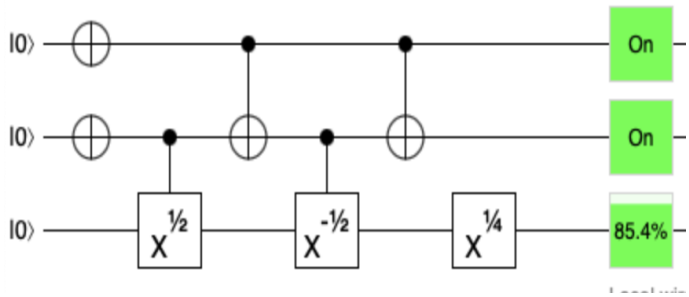


Thus, all behavior is desired except in the first case – we don't want a 50% chance of the last qubit turning in the default case. Thus, this is where we must sacrifice accuracy for the sake of approximation – we replace $X^{1/2}$ with $X^{1/4}$ to give us a better chance of getting 0 for the last qubit in the base case (because $X^{1/4}$ measures ON about 15% of the time compared to $X^{1/2}$ which measures ON 50% of the time). So, we have the following circuit/behavior:


|0⟩
|0⟩
|0⟩
X ½
X -½
X ¼
Off
Off
14.6%


|0⟩
|0⟩
|0⟩
X ½
X -½
X ¼
On
Off
14.6%


|0⟩
|0⟩
|0⟩
X ½
X -½
X ¼
Off
On
14.6%


|0⟩
|0⟩
|0⟩
X ½
X -½
X ¼
On
On
85.4%

Thus, in terms of probabilities, the above circuit has a favorable output – with the first two qubits being $|00\rangle$, we get a low chance of the last qubit being $|1\rangle$; with one of the first two qubits being ON, we get a low chance of the last qubit being ON; finally, with the top two qubits being $|11\rangle$, we get a high chance of the last qubit being $|1\rangle$. Thus, we compute the Hilbert-Schmidt distance between the above circuit and $CCNOT$ and see what the result is. We use the following python script to compute the distance:

```
# Computing square root of NOT and inverse of square root of NOT
evalues, evectors = np.linalg.eig(X)
sqrt_X = evectors * np.sqrt(evalues) @ np.linalg.inv(evectors)
negative_sqrt_X = np.linalg.inv(sqrt_X)

# Computing fourth root of NOT
evalues, evectors = np.linalg.eig(sqrt_X)
fourth_root_X = evectors * np.sqrt(evalues) @ np.linalg.inv(evectors)

C_sqrt_X = tensor_product(outer_prod_1, sqrt_X) + tensor_product(outer_prod_0, I)
C_negative_sqrtX = tensor_product(outer_prod_1, negative_sqrt_X) + tensor_product(outer_prod_0, I)

#computing the Toffoli approximation:
comp_one = tensor_product(I, C_sqrt_X)
comp_two = tensor_product(CX, I)
comp_three = tensor_product(I, C_negative_sqrtX)
comp_four = tensor_product(CX, I)
comp_five = tensor_product(tensor_product(I, I), fourth_root_X)

V = comp_one.dot(comp_two.dot(comp_three.dot(comp_four.dot(comp_five))))

print(hilbert_schmidt_distance_CCNOT(V))
```

The above script has the following definitions:

```
CCNOT = np.array([
    [1 + 0j, 0 + 0j, 0 + 0j,  0 + 0j,  0 + 0j, 0 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 1 + 0j, 0 + 0j,  0 + 0j,  0 + 0j, 0 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 0 + 0j, 1 + 0j,  0 + 0j,  0 + 0j, 0 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 0 + 0j, 0 + 0j,  1 + 0j,  0 + 0j, 0 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 0 + 0j, 0 + 0j,  0 + 0j,  1 + 0j, 0 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 0 + 0j, 0 + 0j,  0 + 0j,  0 + 0j, 1 + 0j, 0 + 0j, 0 + 0j ],
    [0 + 0j, 0 + 0j, 0 + 0j,  0 + 0j,  0 + 0j, 0 + 0j, 0 + 0j, 1 + 0j ],
    [0 + 0j, 0 + 0j, 0 + 0j,  0 + 0j,  0 + 0j, 0 + 0j, 1 + 0j, 0 + 0j  ],
])

outer_prod_0 = np.array([
    [1 + 0j, 0 + 0j],
    [0 + 0j, 0 + 0j]
])

outer_prod_1 = np.array([
    [0 + 0j, 0 + 0j],
    [0 + 0j, 1 + 0j]
])


I = np.array([
    [1 + 0j, 0 + 0j],
    [0 + 0j, 1 + 0j]
])

X = np.array([
    [0 + 0j, 1 + 0j],
    [1 + 0j, 0 + 0j]
])

CX = np.array([
    [1 + 0j, 0 + 0j, 0 + 0j, 0 + 0j],
    [0 + 0j, 1 + 0j, 0 + 0j, 0 + 0j],
    [0 + 0j, 0 + 0j, 0 + 0j, 1 + 0j],
    [0 + 0j, 0 + 0j, 1 + 0j, 0 + 0j],
])

T = np.array([
    [1 + 0j, 0 + 0j],
    [0 + 0j, 1/(math.sqrt(2)) + 1j/(math.sqrt(2))]
])
```

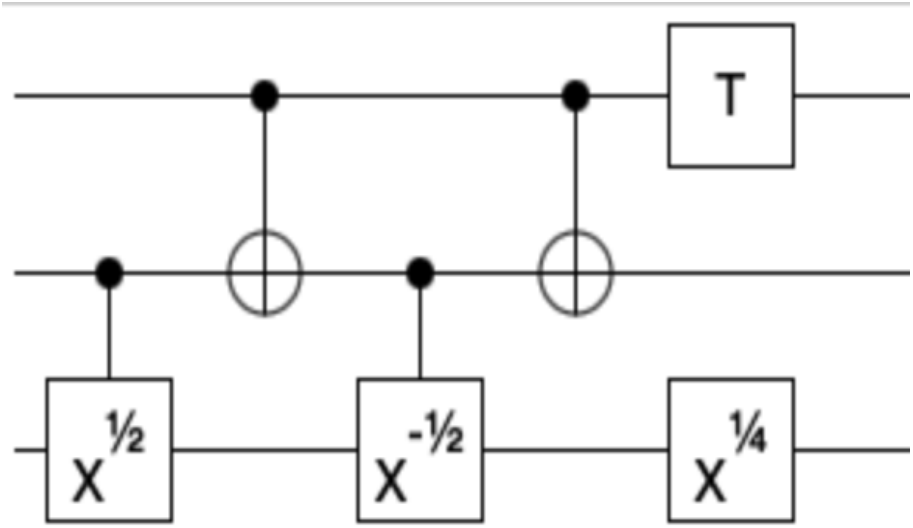
tensor_product and *hilbert_schmidt_distance_CCNOT* are the following helper functions:

```
#helper function for computing tensor products
def tensor_product(A, B):
    #Input:  matrix A and matrix B (square matricies)
    #Output: tensor product of A and B
    tensor_product = np.zeros((len(A)*len(B), len(A[0])*len(B[0])), dtype=np.complex_)
    for i in range(len(A)):
        for j in range(len(A[0])):
            a = A[i][j]
            a_x_B = a*B
            for m in range(len(B)):
                for n in range(len(B[0])):
                    tensor_product[i*len(B) + m][j*len(B[0]) + n] = a_x_B[m][n]
    return tensor_product

#returns the hilbert-schmidt distance with CCNOT
def hilbert_schmidt_distance_CCNOT(V):
    norm_sq_tr_U_dag_V = abs(np.trace(CCNOT.dot(V)))**2
    return math.sqrt(1 - norm_sq_tr_U_dag_V/(64))
```

The output we get with our current approximation of Tofolli is:

```
(base) user@Rishis-MacBook-Pro Toffoli Approximation % python3 toffoli_approx.py
0.5210053832799874
```

which is not the desired 0.382. Because the distance is high despite the probabilities closely matching Toffoli, the issue likely lies in high phase differences between our approximate Toffoli and the actual Toffoli. As inspiration from Professor Palsberg's implementation of Toffoli, we add a T gate to the first qubit of our circuit (which influences the phase only); we thus have the following circuit for our approximation for Toffoli:



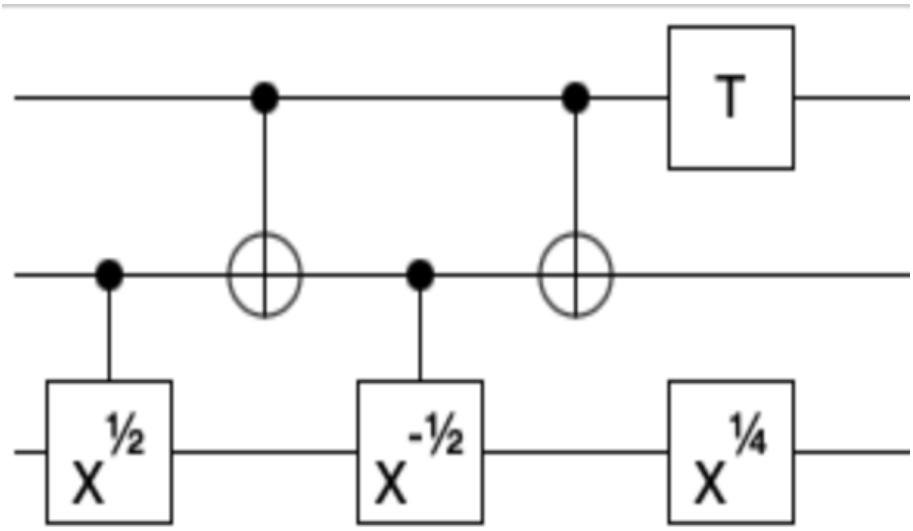
All the probabilities of the above circuit are the same as without the T gate because T only influences the phase. We thus update our script:

```
# Computing square root of NOT and inverse of square root of NOT
evalues, evectors = np.linalg.eig(X)
sqrt_X = evectors * np.sqrt(evalues) @ np.linalg.inv(evectors)
negative_sqrt_X = np.linalg.inv(sqrt_X)

# Computing fourth root of NOT
evalues, evectors = np.linalg.eig(sqrt_X)
fourth_root_X = evectors * np.sqrt(evalues) @ np.linalg.inv(evectors)

C_sqrt_X = tensor_product(outer_prod_1, sqrt_X) + tensor_product(outer_prod_0, I)
C_negative_sqrtX = tensor_product(outer_prod_1, negative_sqrt_X) + tensor_product(outer_prod_0, I)

#computing the Toffoli approximation:
comp_one = tensor_product(I, C_sqrt_X)
comp_two = tensor_product(CX, I)
comp_three = tensor_product(I, C_negative_sqrtX)
comp_four = tensor_product(CX, I)
comp_five = tensor_product(tensor_product(T, I), fourth_root_X)

V = comp_one.dot(comp_two.dot(comp_three.dot(comp_four.dot(comp_five))))

print(hilbert_schmidt_distance_CCNOT(V))
```

Specifically, we changed *comp_five* to be the product $T \otimes I \otimes X^{1/4}$ to reflect our circuit. The resulting Hilbert-Schmidt distance is

```
(base) user@Rishis-MacBook-Pro Toffoli Approximation % python3 toffoli_approx.py
0.3826834323650904
```

Which is precisely the distance we desire. Thus, our final circuit for approximating Toffoli is



and has a Hilbert-Schmidt distance of 0.38268 from the actual Toffoli.