Decomposition of three qubit unitaries
Balint Pato
2020

Abstract

We would like to implement arbitrary three-qubit unitary decomposition into two-qubit and one-qubit gates in Cirq (https://github.com/quantumlib/Cirq/issues/451). This write-up explores the methodology described in [SBM06] which is the best known algorithm in terms of CNOT count, giving 20 CNOT decomposition for any three qubit unitaries.

1 Lower Bounds

Decomposing a unitary in the general case results in an exponential amount of single qubit and two-qubit gates. Based on [SMB04] the theoretical lower bound for `CNOT`s in three qubit circuits are $\frac{1}{4}(4^n - 3n - 1) = 54/4 = 13.5 \simeq 14$ in the gateset $\{R_x, R_z, CNOT\}$.

The following algorithm gives 20 `CNOT`s.

2 Quantum multiplexors

The idea of quantum multiplexors is described in [SBM06]:

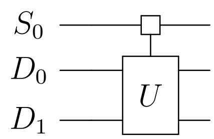
A quantum multiplexor is a unitary that leaves the S, *select qubits* in their original state while changes the *data qubits* depending on the value of S. For each possible classic value of S the multiplexor can act with a different unitary.

While this sounds like controlled gates - they are not. This is a generalization of the control notion, where based on S different unitaries can be executed, in contrast controlled gates are only "choosing" between identity or a single unitary.

Notation: An n-qubit multiplexor U in circuit diagrams is denoted by "$\square$" on each select qubit, connected by a vertical line to a gate on the remaining data qubits, with the U symbol on the rectangle over the *data qubits*. While technically U denotes the whole unitary and not just the $n_{\text{data qubits}}$ qubit unitary that the actual "box" covers, this circuit notation allows to express the multiplexor structure.

Examples:

- S is the most significant qubit - this case U is block diagonal: $U = \begin{pmatrix} U_0 & \\ & U_1 \end{pmatrix}$, which we can denote with $U = U_0 \oplus U_1$



- CNOT $= I \oplus \sigma^x$

3 Shannon decomposition

The classical Shannon expansion of boolean functions is an important result that describes an N variable boolean function as the XOR of two $N-1$ variable Boolean functions that are both restricted in one variable to 0 or 1 respectively:

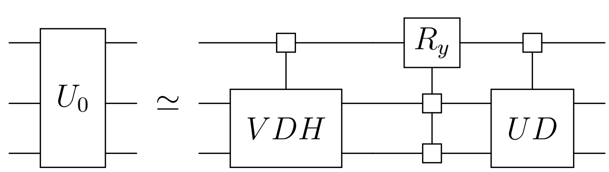
$$f(x_1, x_2, .., x_N) = (x_1 \wedge f(x_1 = 0, x_2, x_3, ..., x_N)) \oplus (\neg x_1 \wedge f_1(x_1 = 1, x_2, x_3, ..., x_N))$$

We can start to see how the language of quantum multiplexors will be helpful to express this kind of decomposition of step-by-step smaller qubit count operators.

3.1 The unoptimized algorithm

We start with an unoptimized algorithm, that gives 24 `CNOT`s. We will optimize it later to achieve the promised 20 `CNOT` count.

- **Step 1:** Cosine Sine decomposition (Theorem 10)



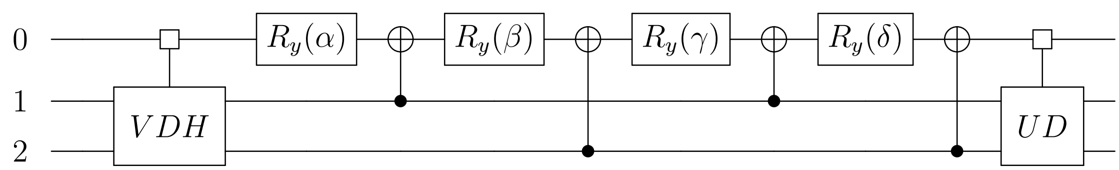
Where $UD = \begin{pmatrix} u_1 & 0 \\ 0 & u_2 \end{pmatrix} \in SU(8)$ is a multiplexor between u_1 and u_2 and similarly $VDH = \begin{pmatrix} v_{1h} & 0 \\ 0 & v_{2h} \end{pmatrix} \in SU(8)$ and $CS = \begin{pmatrix} C & -S \\ S & C \end{pmatrix} \in SU(8)$, where C and S are

4x4 diagonal matrices that satisfy $C^2 + S^2 = I$, finally $\theta_i, i \in \{0, 1, 2, 3\}$ are the four possible rotations of the R_y rotation that is being multiplexed by the four possible states of the two least significant bits.

Since SciPy version 1.5.0 the cossin method implements the Cosine Sine decomposition.

```
from scipy.linalg import cossin
(u1, u2), theta, (v1h, v2h) = cossin(U, 4, 4, separate=True)
```

- **Step 2:** demultiplex the multiplexed R_y: it's easy to verify that the following circuit satisfies the required multiplexing logic, that is $|k> \rightarrow R_y(\theta_k)$:



, where

$$\begin{aligned}
\alpha &= \theta_0 + \theta_1 + \theta_2 + \theta_3 \\
\beta &= \theta_0 + \theta_1 - \theta_2 - \theta_3 \\
\gamma &= \theta_0 - \theta_1 - \theta_2 + \theta_3 \\
\delta &= \theta_0 - \theta_1 + \theta_2 - \theta_3
\end{aligned}$$

- **Step 3:** demultiplex the two-qubit multiplexor UD:a block diagonal matrix can be diagonalized in a way that creates two two-qubit gates and a multiplexed R_z in the middle multiplexed on qubits 0 and 1, acting on qubit 2, i.e $u_1 \oplus u_2 = (I \otimes V)(D \oplus D^\dagger)(I \otimes W)$:

$$\begin{pmatrix} u_1 & \\ & u_2 \end{pmatrix} = \begin{pmatrix} V & \\ & V \end{pmatrix} \begin{pmatrix} D & \\ & D^\dagger \end{pmatrix} \begin{pmatrix} W & \\ & W \end{pmatrix}$$

The calculation of V, D and W comes from:

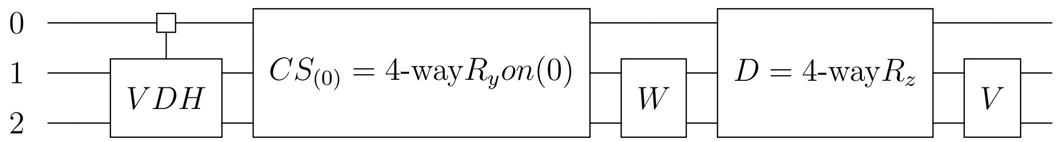
$$\begin{aligned}
u_1 &= VDW \\
u_2 &= VD^\dagger W \\
u_2^\dagger &= W^\dagger D V^\dagger \\
u_1 u_2^\dagger &= VDWW^\dagger DV^\dagger = VD^2V^\dagger
\end{aligned}$$

Where D is diagonal. We implemented `cirq.unitary_eig` to ensure that the resulting eigenvectors are orthogonal - i.e the resulting V is unitary:

```
u1u2 = u1 @ u2.conj().T
eigvals, V = cirq.unitary_eig(u1u2)
d = np.diag(np.sqrt(eigvals))
```

W can be easily expressed as $W = DV^\dagger u_2$.

- **Step 4:** Implementing the Diagonal D is very similar to CS, using a 4-way R_z gate. At this point we have CS implemented as a four-way R_y gate, W and V two-qubit unitaries, D implemented as a four-way R_z gate.



- **Step 5:** similarly decompose VDH - giving 4 `CNOT`s for the 4-way multiplexed R_z

- **Step 6:** decompose the four two qubit operators (using the KAK decomposition) that gives 3 `CNOT`s for each operator

This gives 24 `CNOT`s = 4 x two-qubit operators x 3 `CNOT`s (KAK) + 2 x 4-way multiplexed R_z gates x 4 `CNOT`s (VDH and UD diagonals) + 1 x 4-way multiplexed R_y gate x 4 `CNOT`s in the middle (CS)

3.2 Optimizations

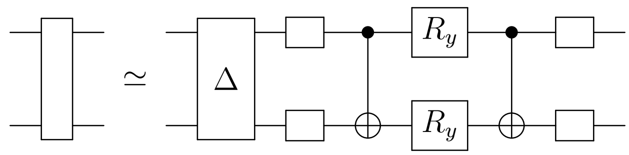
3.2.1 `CNOT` → `CZ` in R_y

Appendix A.1 in [SBM06] explains how replacing the `CNOT`s with `CZ`s works equivalently in the multiplexed R_y implementation. The terminal CZ, as the CZ gate is diagonal, can be merged with the neighboring generic two-qubit multiplexer (UD).

Now, we are down to 3 `CZ`s and 20`CNOT`s.

3.2.2 Eager diagonals

Based on Theorem 14 in Appendix A.2 by [SBM06] there exists a diagonal gate Δ that we can extract from any two-qubit unitary that will leave a two-qubit gate that can be decomposed with only two `CNOT` gates:



This diagonal commutes through the controls of the multiplexed R_y and can be merged with the generic two-qubit multiplexers on the left of the circuit. As the KAK decomposition implemented in Cirq recognizes the two-CNOT circuits (is this true in general?), as long as we extract this diagonal, we can win 3 more CNOTs.

In order to understand this decomposition we need to look at [MBS03] for more details.

4 Extracting a diagonal from two-qubit circuits

4.1 Invariants of two-qubit unitaries

[MBS03] describes equivalence classes of two qubit special unitaries depending on whether they require zero, one, two or three CNOTs to implement them.

$U(4)$ is the group of two-qubit unitaries, $SU(4)$ is the group of *determinant one* two-qubit unitaries, the special unitary group.

Def: $\gamma : U(4) \to U(4), \gamma(u) = u(\sigma^y)^{\otimes 2} u^T (\sigma^y)^{\otimes 2}$

Def: $\chi(u)(x) = p(x) = det(xI - u)$ the characteristic polynomial.

Def: u and v two are *equivalent up to local unitaries* if there exist one-qubit operators that, when pre- and post-composing with u, up to a global phase we get v. Denoted by $u \equiv v$.

[MBS03], **Proposition II.1**: $\forall u, v \in SU(4) u \equiv v \iff \chi(\gamma(u)) = \chi(\pm\gamma(v))$

[MBS03], **Proposition III.1**: An operator $u \in SU(4)$ can be simulated with 0 CNOT gates, if $\chi(\gamma(u)) = (x+1)^4$ or $(x-1)^4$

[MBS03], **Proposition III.2**: An operator $u \in SU(4)$ can be simulated with 1 CNOT gate, if $\chi(\gamma(u)) = (x+i)^2(x-i)^2$

[MBS03], **Proposition III.3**: An operator $u \in SU(4)$ can be simulated with 2 CNOT gates, if $\chi(\gamma(u))$ has real coefficients, which occurs if $tr[\gamma(u)] \in \mathcal{R}$

4.2 Extracting the diagonal

The diagonal extraction is presented by in another paper by the same authors in [SMB04]. C_2^1 represents a CNOT gate with control on the 1st qubit, target on the 2nd qubit:

[SMB04], **Proposition V.2**: For any $u \in SU(4)$ one can find θ, ϕ, ψ so that $\chi[\gamma(uC_2^1(I \otimes R_z(\psi))C_2^1] = \chi[\gamma(C_2^1(R_x(\theta) \otimes R_z(\phi))C_2^1]$.

Which is exactly what we want: compose our two-qubit unitary (u) from the left (in the circuit diagram) with a diagonal ($C_2^1(I \otimes R_z(\psi))C_2^1$ is diagonal) to get a unitary that can be implemented with only two CNOTs.

The proof is constructive and contains the algorithm to find ψ, however **it does have a typo/bug in the formulae**:

$$\Delta := C_2^1(I \otimes R_z(\psi))C_2^1$$
$$tr[\gamma(u\Delta)] = (t_1 + t_4)e^{-i\psi} + (t_2 + t_3)e^{i\psi}$$

, where t_1, t_2, t_3, t_4 are the diagonal entries of $\gamma(u^T)^T$.

Now, the paper claims that *"We may ensure that this number is real by setting* $tan(\psi) = \frac{Im(t_1+t_2+t_3+t_4)}{Re(t_1+t_2-t_3-t_4)}$".

Which is incorrect, it is relatively easy to deduce that $tan(\psi) = \frac{Im(t_1+t_2+t_3+t_4)}{Re(t_1+t_4-t_2-t_3)}$ is the right formula.

Also, there is one missing case mentioned in the paper:

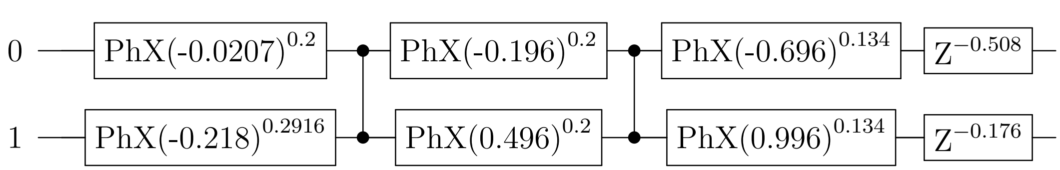
- when $Re(t_1 + t_4 - t_2 - t_3) = 0$ and will mean that:

$$(t_1 + t_4)e^{-i\psi} + (t_2 + t_3)e^{i\psi} \in \mathbb{R} \implies$$
$$e^{i\psi} = -e^{-i\psi}$$
$$\psi = 3\pi/2 \text{ or } \pi/2$$

The python code:

```
def special(u):
    return u / (np.linalg.det(u) ** (1 / 4))
```

```
def g(u):
    yy = np.kron(cirq.Y._unitary_(), cirq.Y._unitary_())
    return u @ yy @ u.T @ yy


def extract_right_diag(U):
    u = special(U)
    t = _gamma(_to_special(U).T).T.diagonal()
    k = np.real(t[0] + t[3] - t[1] - t[2])

    if k == 0:
        # in the end we have to pick a psi that makes sure that
        # exp(-i*psi) (t[0]+t[3]) + exp(i*psi) (t[1]+t[2]) is real
        # both pi/2 or 3pi/2 can work
        psi = np.pi/2
    else:
        psi = np.arctan(np.imag(np.sum(t)) / k)

    a, b = cirq.LineQubit.range(2)
    c_d = cirq.Circuit([cirq.CNOT(a, b), cirq.rz(psi)(b), cirq.CNOT(a, b)])
    return c_d._unitary_()


V = circuit._unitary_()
dV = extract_right_diag(V)
V = V @ dV
print(cirq.Circuit(
        cirq.optimizers.two_qubit_matrix_to_operations(
            a,b,V,allow_partial_czs=False
        )))
np.trace(g(special(V)))

...

(-2.618033988749896-2.7755575615628914e-16j)
```



References

[MBS03] Igor L Markov, Stephen S Bullock, and Vivek V Shende. "Recognizing Small-Circuit Structure in Two-Qubit Operators and Timing Hamiltonians to Compute Controlled-Not Gates". In: *Quant-Ph/0308045* (2003), pp. 3–6. DOI: doi:10.1103/PhysRevA.70.012310. arXiv: 0308045 [quant-ph]. URL: http://arxiv.org/abs/quant-ph/0308045%7B%5C%%7D5Cnhttp://www.arxiv.org/pdf/quant-ph/0308045.pdf.

[SMB04] Vivek V Shende, Igor L Markov, and Stephen S Bullock. *Minimal Universal Two-Qubit CNOT-based Circuits*. Tech. rep. 2004.

[SBM06] Vivek V Shende, Stephen S Bullock, and Igor L Markov. *Synthesis of Quantum Logic Circuits*. Tech. rep. 2006.