Quantum Circuits for State Permutations using Routing via Matchings and Multiplexed-Rx

Yao Tang, Pablo André-Martínez, Silas Dilkes

QUANTINUUM

Contents

State Permutation Problem

Given a permutation of basis states, implement the permutation using a quantum circuit.

Also known as *Boolean Reversible Function Synthesis*

State Permutation Problem

Given a permutation of basis states, implement the permutation using a quantum circuit.

Also known as *Boolean Reversible Function Synthesis*

E.g. Given permutation `|00>:|11>, |01>:|10>, |10>:|01>, |11>:|00>`

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle$$

State Permutation Problem

Given a permutation of basis states, implement the permutation using a quantum circuit.

Also known as *Boolean Reversible Function Synthesis*

E.g. Given permutation `|00>:|11>, |01>:|10>, |10>:|01>, |11>:|00>`

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle$$

permute

$$|\psi'\rangle = \alpha_0 |11\rangle + \alpha_1 |10\rangle + \alpha_2 |01\rangle + \alpha_3 |00\rangle$$

Naive Approach

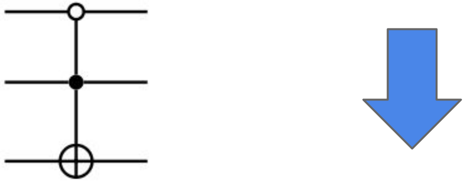
1. Traverse the basis states following the Gray code.

   000, 001, 011, 010, 110,...

2. Bubble sort the states.
3. Each **swap** is implemented using a Multi-controlled Toffoli gate.

E.g.

$$|\psi\rangle = \cdots \alpha\,|010\rangle + \beta\,|011\rangle + \cdots$$



Naive Approach

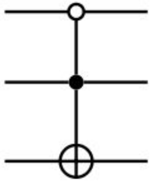
1. Traverse the basis states following the Gray code.

   000, 001, 011, 010, 110,...

2. Bubble sort the states.
3. Each **swap** is implemented using a Multi-controlled Toffoli gate.

E.g.

$$|\psi\rangle = \cdots \alpha |010\rangle + \beta |011\rangle + \cdots$$



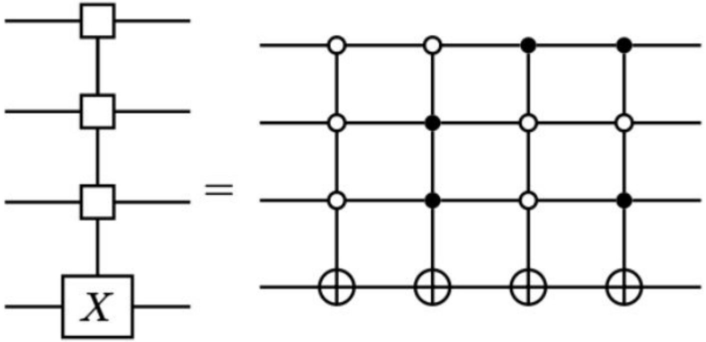
$$|\psi'\rangle = \cdots \alpha |011\rangle + \beta |010\rangle + \cdots$$

Multiplexed-X

Def: $$\text{Multiplexed-X} = \sum_i |i\rangle\langle i| \otimes U_i$$

$$U_i \in \{X, I\}$$

Special case of multiplexor - apply either I or X to the target qubit according to the bitstring on the control qubits.



Also known as ***Single-target Boolean Function***

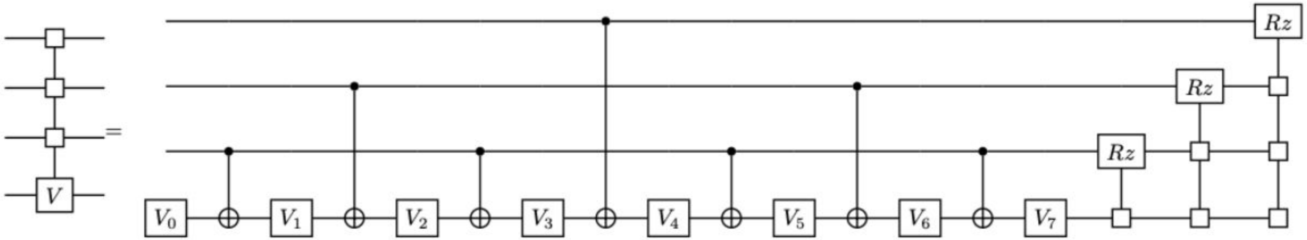
Multiplexed-X

Def:

$$\text{Multiplexed-X} = \sum_i |i\rangle \langle i| \otimes U_i$$

$$U_i \in \{X, I\}$$

Special case of multiplexor - apply either I or X to the target qubit according to the bitstring on the control qubits.



Multiplexed-X

Def: $$\text{Multiplexed-X} = \sum_i |i\rangle\langle i| \otimes U_i$$

$$U_i \in \{X, I\}$$

Special case of multiplexor - apply either I or X to the target qubit according to the bitstring on the control qubits.

e.g.

$$|\psi\rangle = \alpha_0 |000\rangle + \alpha_1 |001\rangle + \cdots + \alpha_2 |110\rangle + \alpha_3 |111\rangle + \cdots$$

$$\sum_{i \in \{00,11\}} |i\rangle\langle i| \otimes X + \sum_{j \notin \{00,11\}} |j\rangle\langle j| \otimes I$$

Multiplexed-X

Def: $$\text{Multiplexed-X} = \sum_i |i\rangle\langle i| \otimes U_i$$

$$U_i \in \{X, I\}$$

Special case of multiplexor - apply either I or X to the target qubit according to the bitstring on the control qubits.

e.g.

Perform ***multiple swaps*** simultaneously

$$|\psi\rangle = \alpha_0 |000\rangle + \alpha_1 |001\rangle + \cdots + \alpha_2 |110\rangle + \alpha_3 |111\rangle + \cdots$$

$$\sum_{i \in \{00,11\}} |i\rangle\langle i| \otimes X + \sum_{j \notin \{00,11\}} |j\rangle\langle j| \otimes I$$

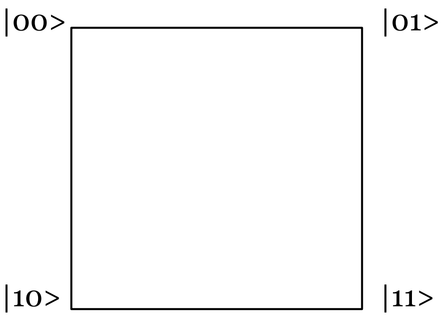
$$|\psi'\rangle = \alpha_0 |001\rangle + \alpha_1 |000\rangle + \cdots + \alpha_2 |111\rangle + \alpha_3 |110\rangle + \cdots$$

Contents

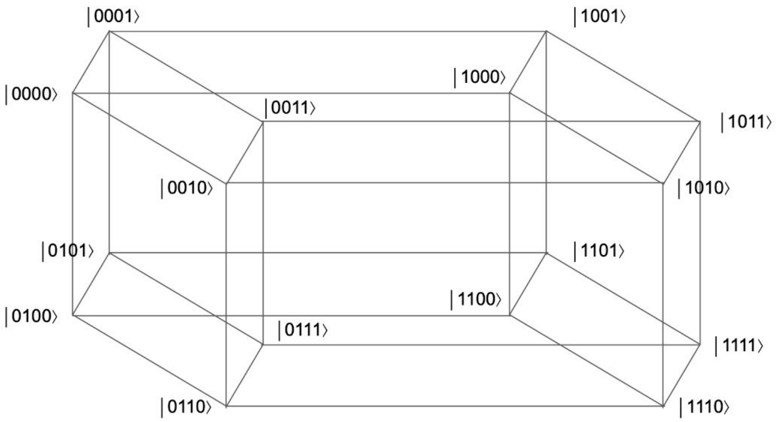
Hypercube Formulation

State graph:

For any n-qubit system, we define a hypercube graph G(V,E), where V is the set all n-qubit basis states, and (u, v) $\in$ E iff Hamming(u, v) = 1.



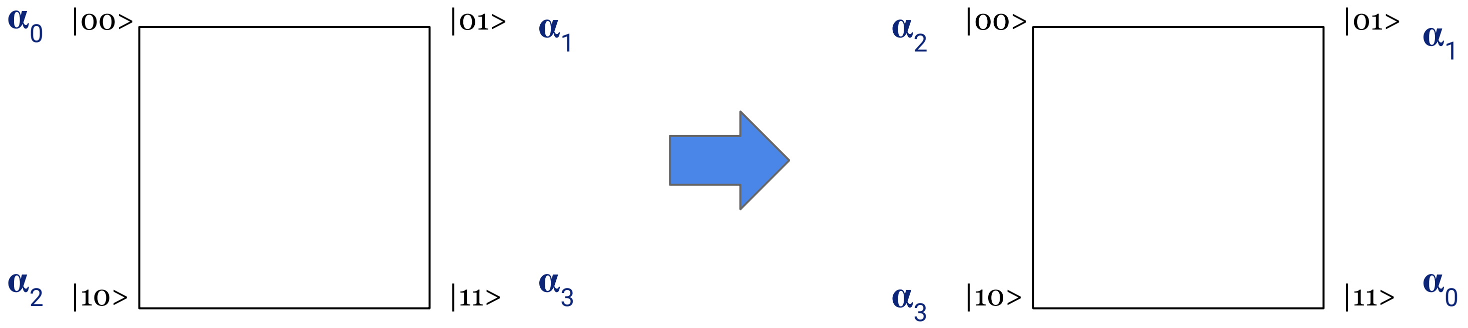
2D cube



4D cube

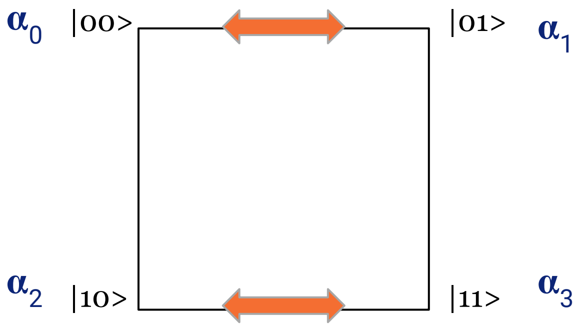
Hypercube Formulation (continued)

Given a state graph, and a set of tokens placed across the graphs vertices, each with a one-to-one correspondence to a destination vertex. The goal is to move the tokens to their respective destinations.



Routing via Matching

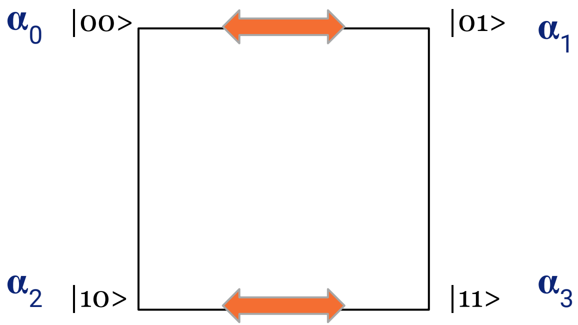
Definition: Swapping the tokens on any set of disjoint edges counts as one operation.



One operation

Routing via Matching

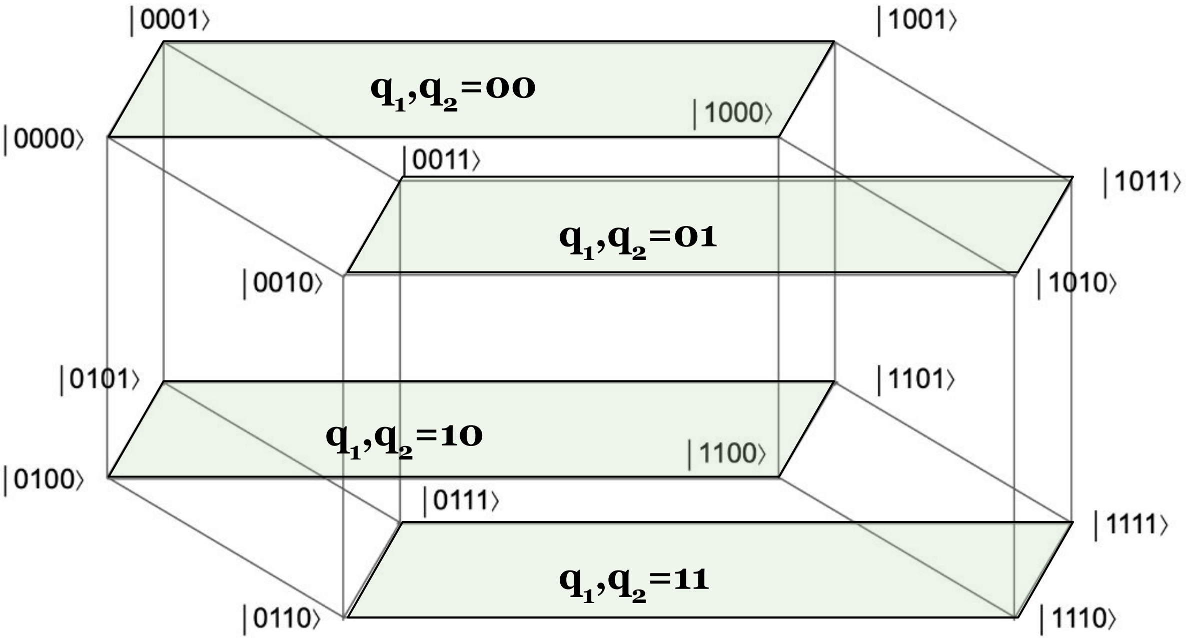
(Alon et al., 1994): Routing via Matching on a hypercube can be done using 2n - 1 operations. Where n is the dimension of the cube, and each operation only involves swaps on **parallel** edges.



We call this **swap along q** because the bitstrings at the endpoints of these edges only differ at **q**

Routing via Matching (algorithm)

Parallel Subcubes:



q_1,q_2 defines four parallel subcubes

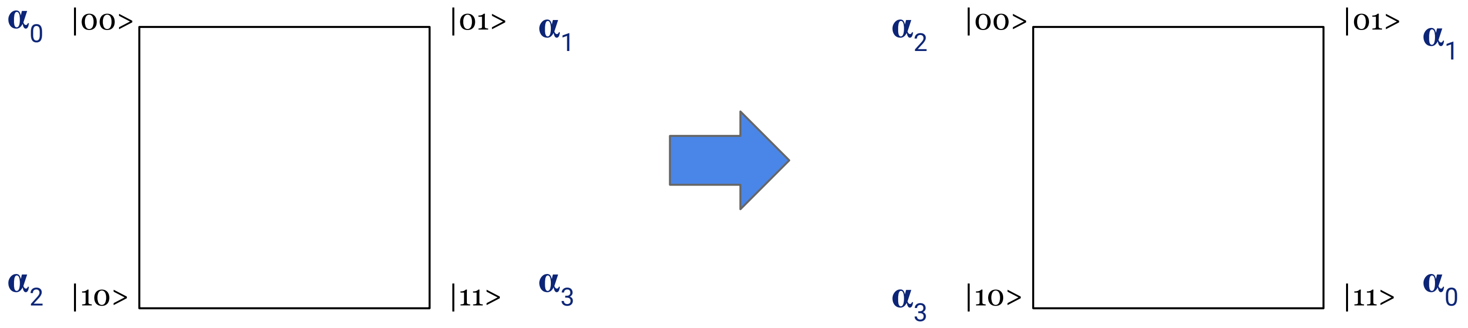
Routing via Matching (algorithm)

For a n-qubit problem, let Q be the set of qubits $\{q_0, q_1, .., q_{n-1}\}$

Algorithm: given a set of parallel subcubes defined by qubits $P \subseteq Q$:

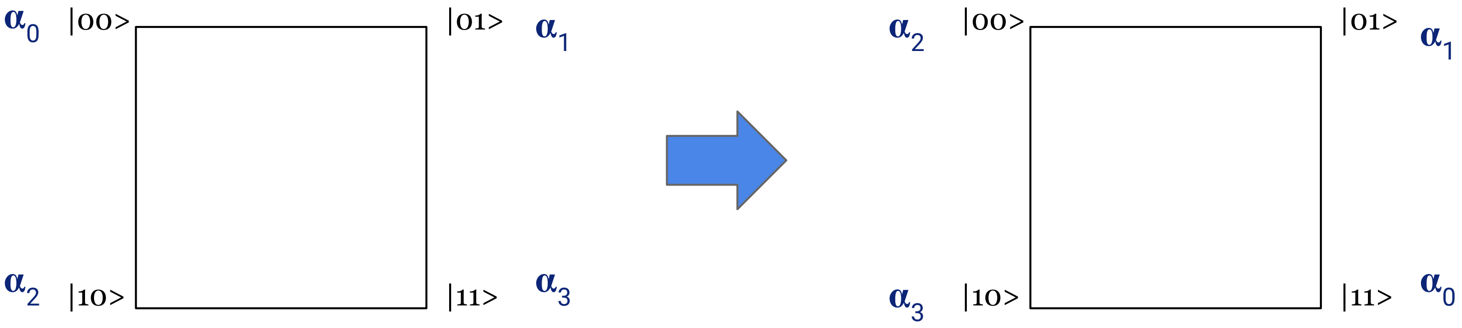
1. Pick a qubit $q \in Q/P$ to further partition each subcube into two parallel subcubes. If $Q/P = \varnothing$, return.
2. For each pair of subcubes obtained from the partitioning, swap tokens along q, such that the destinations of the tokens in each subcube covers all possible bitstrings for $Q/P/\{q\}$. (One Multiplexed-X)
3. Recursively call this function with $P = P \cup \{q\}$.
4. Swap any token with its neighbour along q when its current location and destination doesn't match at bit q. (One Multiplexed-X)

Example



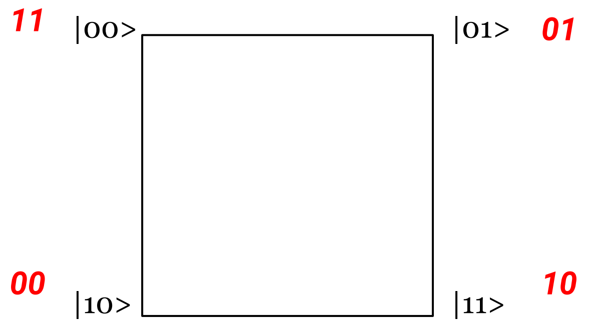
α_0
|00>
|01>
α_1
α_2
|10>
|11>
α_3
α_2
|00>
|01>
α_1
α_3
|10>
|11>
α_0

Example



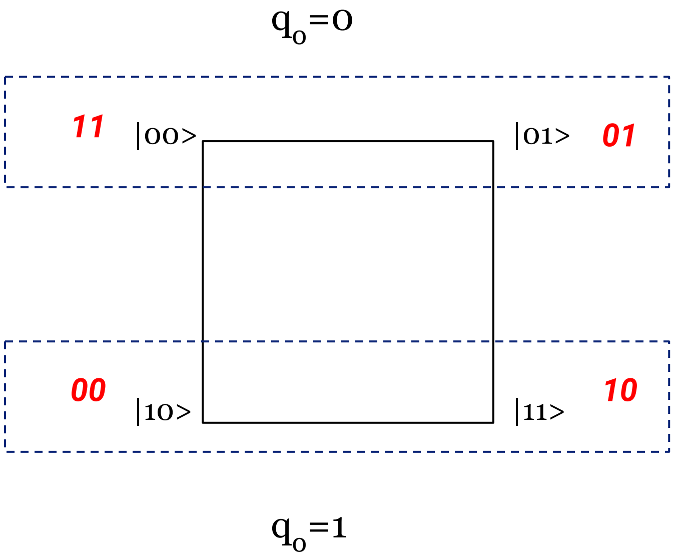
We rename each token with its destination bitstring

Example



Example

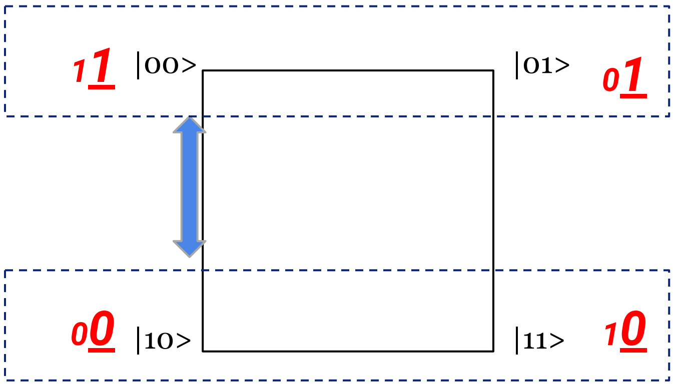
1. Partition along q_0



Example

2. Distribute along q_0

q_0=0



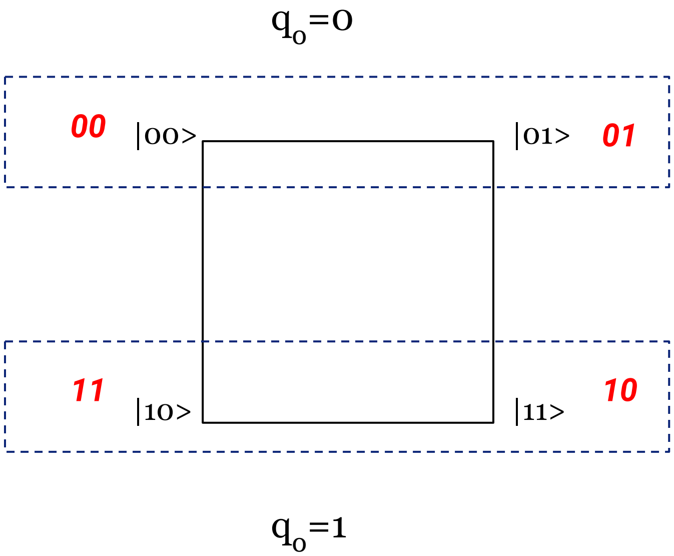
q_0=1

Tokens ***11*** and ***01*** don't cover all possible bitstrings of q_1

This corresponds to a bipartite perfect matching problem, and we use a off-the-shelf Hopcroft-Karp algorithm to solve it.

Example

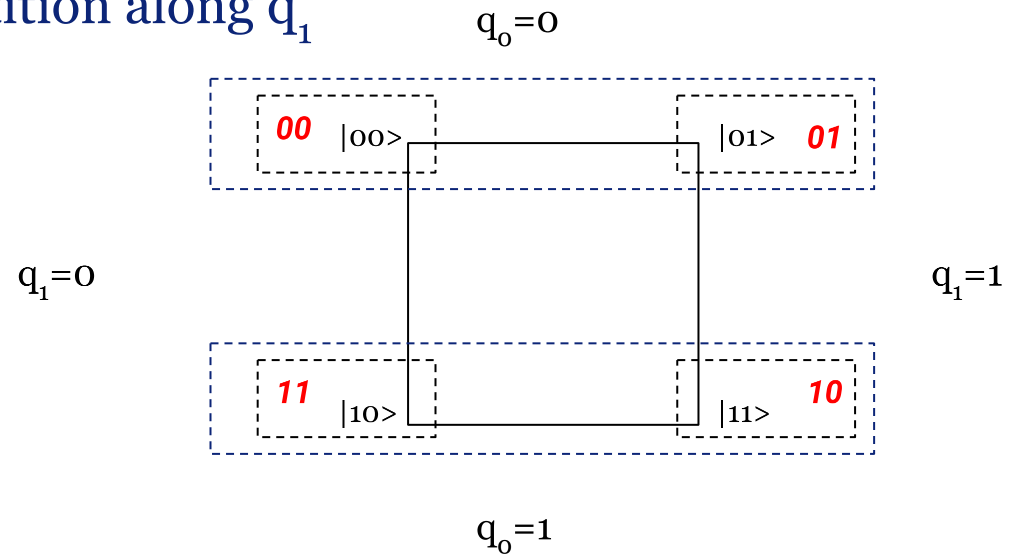
2. Distribute along q_0



Now, tokens ***00*** and ***01*** do cover all possible bitstrings of q_1

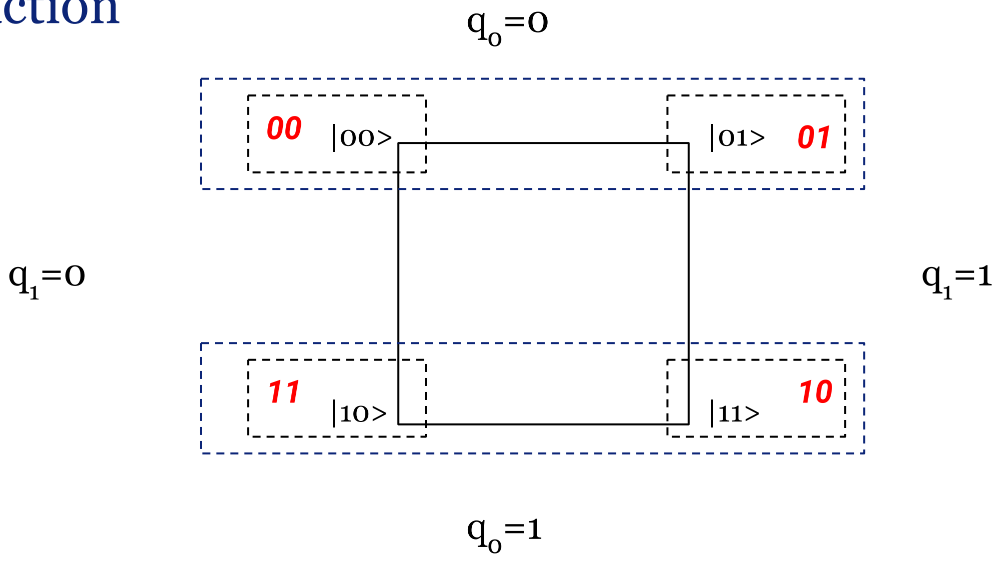
Example

3. Recursive call
    1. Partition along q_1



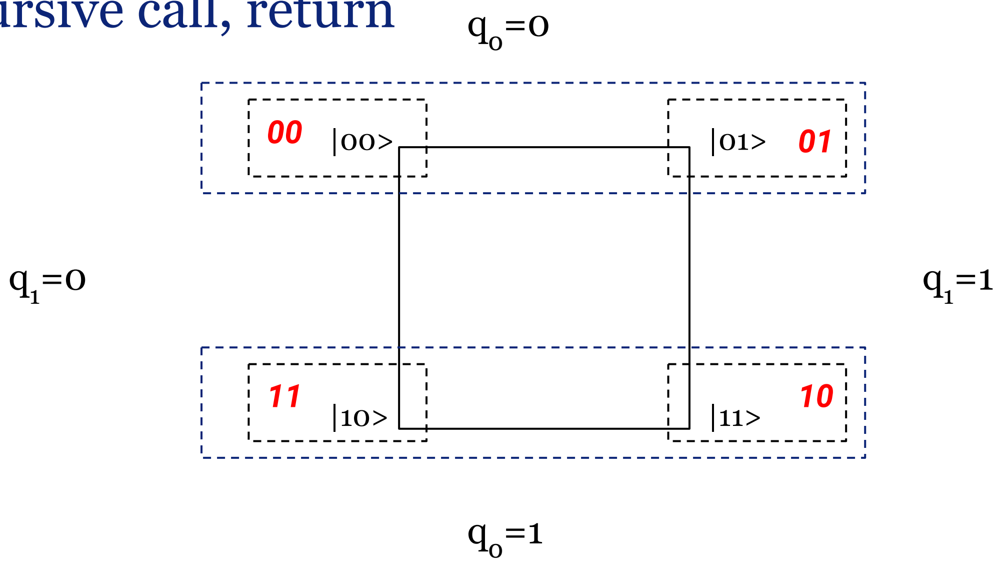
Example

3. Recursive call
    2. No action



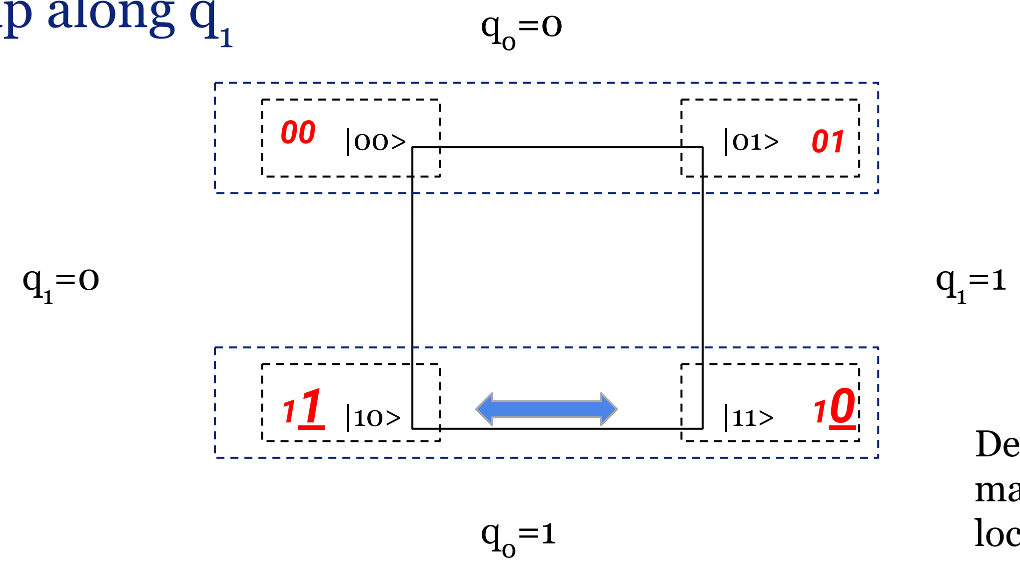
Example

3. Recursive call
    3. Recursive call, return



Example

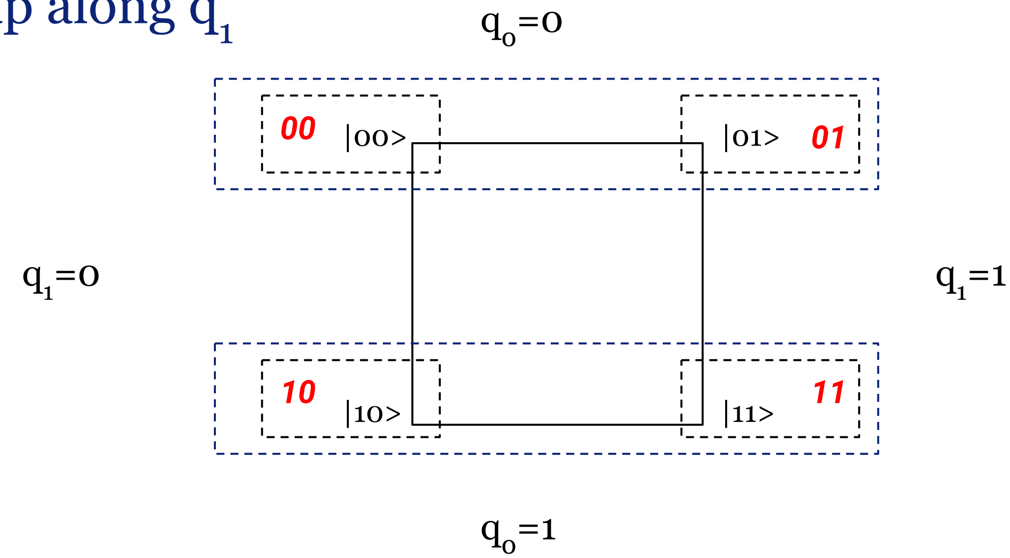
3. Recursive call
    4. Swap along q_1



Destinations doesn't match the current locations at q_1

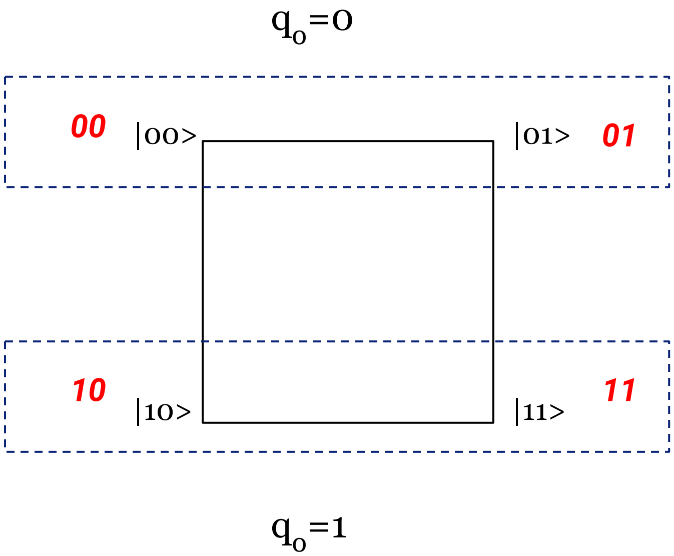
Example

3. Recursive call
4. Swap along q_1



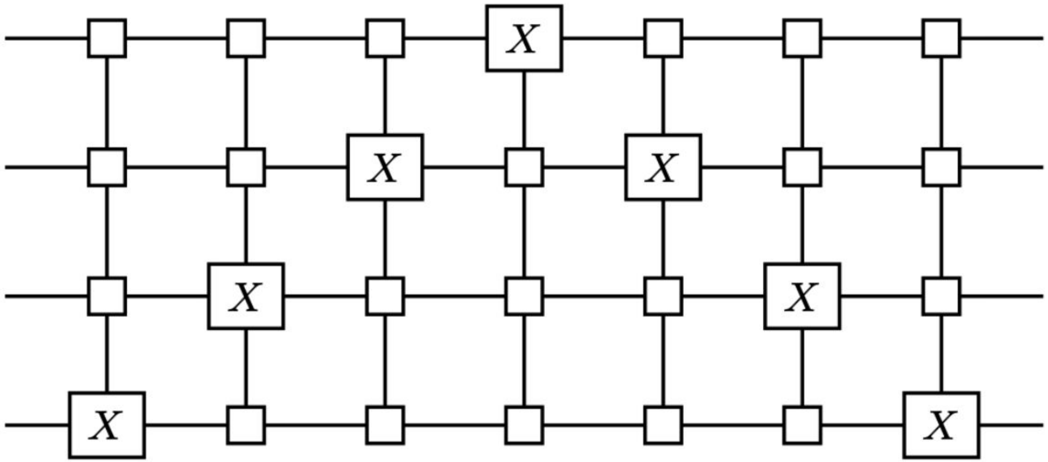
Example

4. Swap along q_0, no action, done



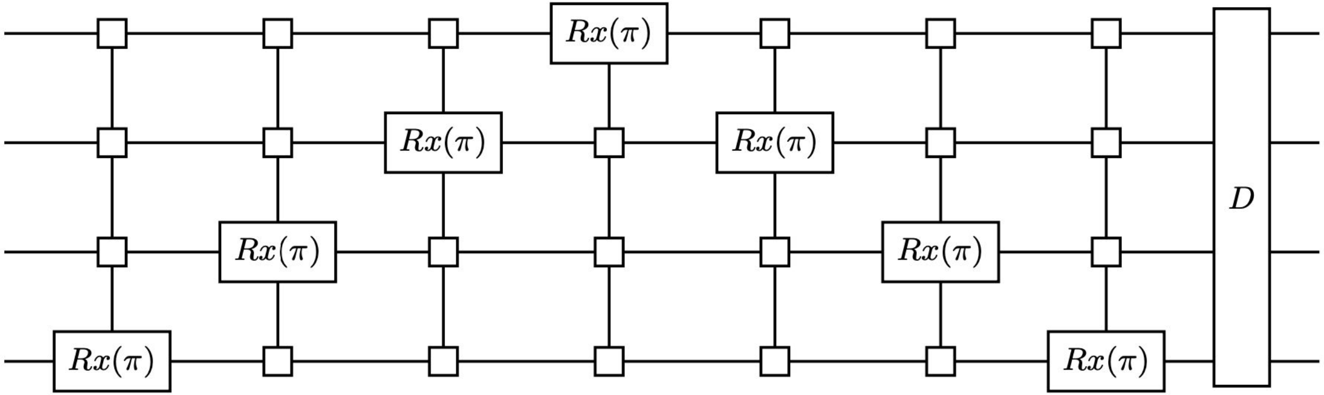
Routing via Matching

We later realised that the circuit produced by our algorithm is equivalent to the Young-subgroup based synthesis method proposed in (Soeken et al., 2019)



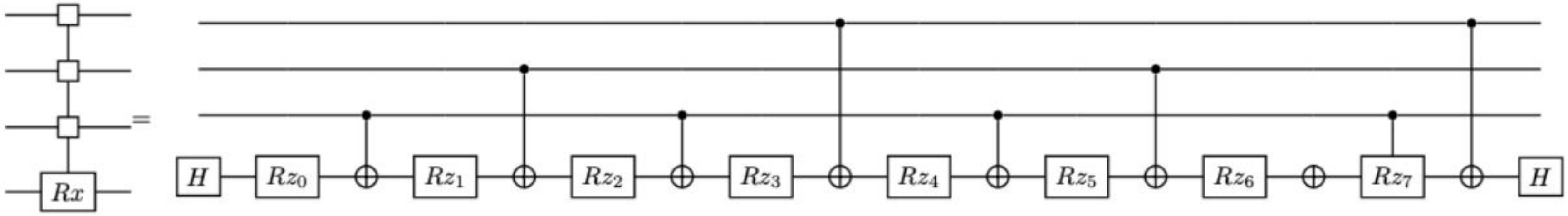
Using Multiplexed-Rx gates

We further improved our approach by replacing the Multiplexed-X gates with Multiplexed-Rx(π) gates. We correct the phase difference introduced by the rotation gates by a diagonal operator at the end, which we decompose as a cascade of Multiplexed-Rz gates.



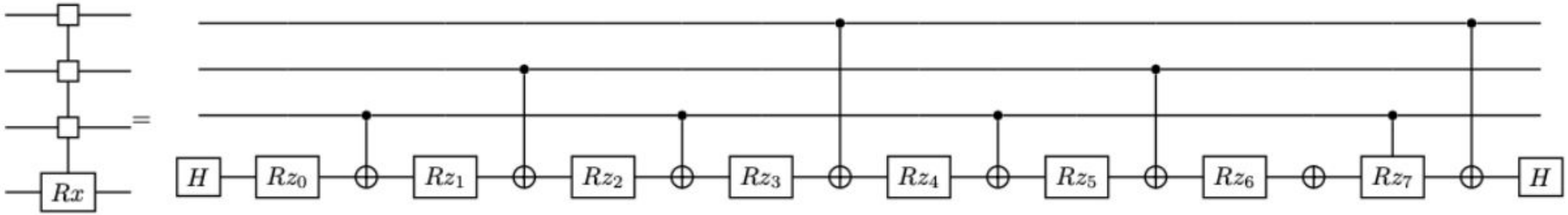
Using Multiplexed-Rx gates

Multiplexed-Rx:

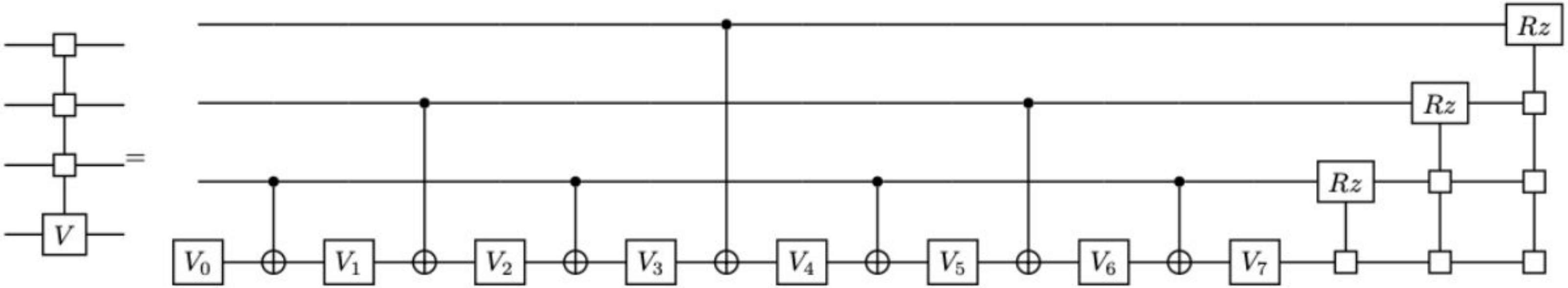


Using Multiplexed-Rx gates

Multiplexed-Rx:



Less expensive than Multiplexed-X



Contents

Results

The proposed method is implemented in TKET as ToffoliBox. We compared the performance of ToffoliBox against (Soeken et al., 2019), which is implemented in Tweedledum.

The benchmark consists of random generated permutations and a set of permutations from the TOF(n), PRIME(n), and HWB(n) families.

We also compared further improvement to CNOT count by applying post-synthesis TKET optimisation passes after synthesis, namely CliffordSimp, SynthesiseTket, and RemoveRedundancies.

Results

#q	SPECTRUM	SPECTRUM + opt	ToffoliBox	ToffoliBox + opt	avg. reduction
3	18.92	15.854	22.82	15.412	-8.47%
4	72.464	70.636	67.916	58.666	14.89%
5	225.992	224.142	172.94	159.858	28.27%
6	613.632	611.372	413.932	397.702	34.85%
7	1536.378	1533.602	957.996	937.642	38.83%

Table 1. Comparison between the average number of CNOT gates produced by TKET and Tweedledum for synthesising circuits for randomly generated permutations. The experiments cover varying numbers of qubits (3 to 7) with 500 gates generated per case. "+ opt" columns show the average CNOT count with post-synthesis optimisation applied. The last column indicates the CNOT reduction rate achieved by ToffoliBox compared to Tweedledum, with optimisation applied.

Results

function	SPECTRUM	SPECTRUM + opt	ToffoliBox	ToffoliBox + opt
tof	6	6	6	6
tof4	14	14	14	14
tof5	30	30	30	30
tof6	62	62	62	62
prime3	20	17	22	13
prime4	58	56	70	51
prime5	182	181	174	167
prime6	520	516	414	378
hwb4	72	72	70	60
hwb5	220	219	174	164

Table 2. Comparison between the number of CNOT gates produced synthesising some special Boolean functions using TKET and Tweedledum.

Further work

- Optimise the order in which you pick qubits
- Use a combination of Rx(π), Rx(-π), Ry(π) and Ry(-π)
- Control logic simplification
- Approximated synthesis methods
- Optimal base case solution (e.g. 3D cube)

Conclusion

- Solving the state permutation problem by treating it as a Routing via Matchings on hypercubes.
  - While this formulation is equivalent to a previously proposed Young-subgroup based synthesis method, our approach might offer a new perspective.
- We introduce the use of Multiplexed-Rx(π) gates for permutation synthesis, resulting in significant reductions in CNOT gate counts.
  - We showed how this outperforms the known state of the art available in Tweedledum for CNOT count.

Thank you!

yao.tang@quantinuum.com