Qubit Recycling Revisited

Analysis, Generalization, and Verification of a Quantum Circuit Transformation

Charles Averill

Quantum Information Science Seminar
The University of Texas at Dallas

October 4, 2024

Background

- *Qubits*, smallest unit of information in a quantum circuit - analogous to bits in classical computing

Background

- *Qubits*, smallest unit of information in a quantum circuit - analogous to bits in classical computing
- Quantum *circuits* pass qubits through *gates* that manipulate their state

Circuit Scale

- Circuits often need to be big to do useful things (crack RSA, simulate quantum systems, solve optimization problems)

Circuit Scale

- Circuits often need to be big to do useful things (crack RSA, simulate quantum systems, solve optimization problems)
- Qubits are really hard to make (decoherence, ability to control state)

Circuit Scale

- Circuits often need to be big to do useful things (crack RSA, simulate quantum systems, solve optimization problems)
- Qubits are really hard to make (decoherence, ability to control state)
- Qubits are really hard to combine into a circuit - the more you add, the more you need (QEC)

Circuit Scale

- Circuits often need to be big to do useful things (crack RSA, simulate quantum systems, solve optimization problems)
- Qubits are really hard to make (decoherence, ability to control state)
- Qubits are really hard to combine into a circuit - the more you add, the more you need (QEC)

$$circuit\ scale \propto implementation\ difficulty$$
$$\Downarrow$$
$$smaller\ scale = better?$$

Smaller Quantum Circuits

There are many ways to reduce the *width*, or number of qubits, of a quantum circuit:

Smaller Quantum Circuits

There are many ways to reduce the *width*, or number of qubits, of a quantum circuit:

- **Gate decomposition** - unfold multi-qubit gates into a sequence of smaller gates

Smaller Quantum Circuits

There are many ways to reduce the *width*, or number of qubits, of a quantum circuit:

- **Gate decomposition** - unfold multi-qubit gates into a sequence of smaller gates
- **Approximation** - trade off accuracy for resource efficiency (Variational Quantum Eigensolvers (VQE) and Quantum Approximate Optimization Algorithm (QAOA) both utilize approximation that results in smaller-than-otherwise circuits)

Smaller Quantum Circuits

There are many ways to reduce the *width*, or number of qubits, of a quantum circuit:

- **Gate decomposition** - unfold multi-qubit gates into a sequence of smaller gates
- **Approximation** - trade off accuracy for resource efficiency (Variational Quantum Eigensolvers (VQE) and Quantum Approximate Optimization Algorithm (QAOA) both utilize approximation that results in smaller-than-otherwise circuits)
- **Recycling** - find valid sequences of *deallocation* (measurement) and *allocation* that allow for the circuit to reuse qubits that it's done operating on

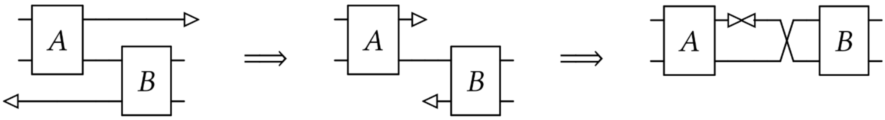
Smaller Quantum Circuits

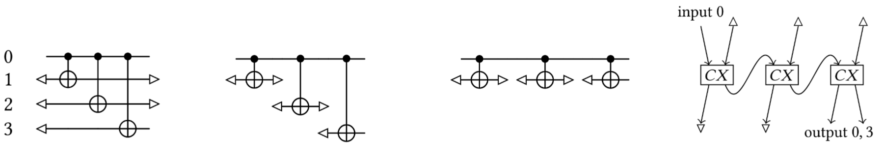
There are many ways to reduce the *width*, or number of qubits, of a quantum circuit:

- **Gate decomposition** - unfold multi-qubit gates into a sequence of smaller gates
- **Approximation** - trade off accuracy for resource efficiency (Variational Quantum Eigensolvers (VQE) and Quantum Approximate Optimization Algorithm (QAOA) both utilize approximation that results in smaller-than-otherwise circuits)
- **Recycling** - find valid sequences of *deallocation* (measurement) and *allocation* that allow for the circuit to reuse qubits that it's done operating on

Let's talk recycling.

Recycling at a High Level





(a) Input circuit. (b) Topological deformation. (c) Renaming and reusing. (d) DAG repr. of Fig. 1a.

Fig. 1. A running example of qubit recycling.

Search Space Size

- The search space for this is really large

Search Space Size

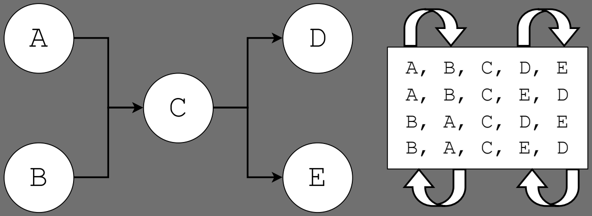
- The search space for this is really large
- Upper bound for topologically identical circuits for a graph with n vertices is $n!$

Search Space Size

- The search space for this is really large
- Upper bound for topologically identical circuits for a graph with n vertices is $n!$
- Even the task of computing the set of identical circuits is #P-complete, so it's even harder than the problems in NP

Search Space Size

- The search space for this is really large
- Upper bound for topologically identical circuits for a graph with n vertices is $n!$
- Even the task of computing the set of identical circuits is #P-complete, so it's even harder than the problems in NP
- #P (Sharp-P) is the class of problems that involves counting the number of ways to solve an NP-class problem. A problem is #P-complete if it is as hard as the hardest problem in #P



Avoiding the Search Space Problem

The author seeks to avoid this problem by looking at the problem from another angle: tackle small *recycling strategies* rather than entire topological orderings.

Avoiding the Search Space Problem

The author seeks to avoid this problem by looking at the problem from another angle: tackle small *recycling strategies* rather than entire topological orderings.

Definition

Recycling Strategy - A map over qubits.
$a \hookrightarrow b$ denotes that qubit b reuses qubit a.

Avoiding the Search Space Problem

The author seeks to avoid this problem by looking at the problem from another angle: tackle small *recycling strategies* rather than entire topological orderings.

Definition

Recycling Strategy - A map over qubits.
$a \hookrightarrow b$ denotes that qubit b reuses qubit a.

- Every qubit recycling solution has a corresponding recycling strategy

Avoiding the Search Space Problem

The author seeks to avoid this problem by looking at the problem from another angle: tackle small *recycling strategies* rather than entire topological orderings.

Definition

Recycling Strategy - A map over qubits.
$a \hookrightarrow b$ denotes that qubit b reuses qubit a.

- Every qubit recycling solution has a corresponding recycling strategy
- Idea: search for the largest recycling strategy (has most qubit reuses) instead of searching for smallest topological ordering

Avoiding the Search Space Problem

The author seeks to avoid this problem by looking at the problem from another angle: tackle small *recycling strategies* rather than entire topological orderings.

Definition

Recycling Strategy - A map over qubits.
$a \hookrightarrow b$ denotes that qubit b reuses qubit a.

- Every qubit recycling solution has a corresponding recycling strategy
- Idea: search for the largest recycling strategy (has most qubit reuses) instead of searching for smallest topological ordering
- The set of valid recycling strategies is much smaller than the set of topological orderings, so this is much more attainable!

Valid Recycling Strategies

Because recycling strategies are just relationships between the qubits of a circuit, we should be able to enumerate all strategies and then check for the valid ones.

Valid Recycling Strategies

Because recycling strategies are just relationships between the qubits of a circuit, we should be able to enumerate all strategies and then check for the valid ones.

Definition

Qubit Dependency Graph (QDG) - a directed graph with qubits as vertices. $a \to b$ denotes that b is *computationally dependent* on the value of a:

- There is a path from the allocation of a to the deallocation of b, or
- a is an input, or b is an output

Valid Recycling Strategies

Because recycling strategies are just relationships between the qubits of a circuit, we should be able to enumerate all strategies and then check for the valid ones.

Definition

Qubit Dependency Graph (QDG) - a directed graph with qubits as vertices. $a \rightarrow b$ denotes that b is *computationally dependent* on the value of a:

- There is a path from the allocation of a to the deallocation of b, or
- a is an input, or b is an output

A recycling strategy is valid if and only if $\rightarrow\hookrightarrow$ is acyclic:

$$(a \rightarrow\hookrightarrow c) \Leftrightarrow (\exists b, a \rightarrow b \wedge b \hookrightarrow c)$$

Valid Recycling Strategies

Because recycling strategies are just relationships between the qubits of a circuit, we should be able to enumerate all strategies and then check for the valid ones.

Definition

Qubit Dependency Graph (QDG) - a directed graph with qubits as vertices. $a \to b$ denotes that b is *computationally dependent* on the value of a:

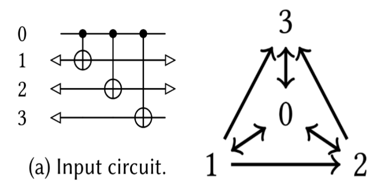
- There is a path from the allocation of a to the deallocation of b, or
- a is an input, or b is an output

A recycling strategy is valid if and only if $\to\hookrightarrow$ is acyclic:

$$(a \to\hookrightarrow c) \Leftrightarrow (\exists b, a \to b \land b \hookrightarrow c)$$

This makes sense - a cycle in $\to\hookrightarrow$ means that our recycling strategy has a circular dependency - not allowed!

Valid Recycling Strategies



Example circuit and corresponding QDG

$\{1 \hookrightarrow 2, 2 \hookrightarrow 3\}$ is a valid recycling strategy because[1]

$$\neg \exists\ a\ b,\ \text{s.t. } a \to b \hookrightarrow a$$

While $\{2 \hookrightarrow 1\}$ is not a valid strategy because of the existence of the cycle

$$1 \to 2 \hookrightarrow 1$$

[1] $a \to b$ here could generate a cycle via $2 \to 0 \to 1$, but $\to\hookrightarrow$ is not acyclic because 0 isn't reusing 2

Largest Recycling Strategy

We've determined which strategies are valid - now we actually need to find the largest one to achieve an optimal solution.

Largest Recycling Strategy

We've determined which strategies are valid - now we actually need to find the largest one to achieve an optimal solution.

- Search space has been reduced, but still very large - larger than $O(n!)$, although this is smaller than the $O(n!)$ of the topological ordering problem, as we have limited the number of qubits to search over

Largest Recycling Strategy

We've determined which strategies are valid - now we actually need to find the largest one to achieve an optimal solution.

- Search space has been reduced, but still very large - larger than $O(n!)$, although this is smaller than the $O(n!)$ of the topological ordering problem, as we have limited the number of qubits to search over
- Brute force is not possible, so

Largest Recycling Strategy

We've determined which strategies are valid - now we actually need to find the largest one to achieve an optimal solution.

- Search space has been reduced, but still very large - larger than $O(n!)$, although this is smaller than the $O(n!)$ of the topological ordering problem, as we have limited the number of qubits to search over
- Brute force is not possible, so
- Can we even find the largest strategy efficiently?

Largest Recycling Strategy

We've determined which strategies are valid - now we actually need to find the largest one to achieve an optimal solution.

- Search space has been reduced, but still very large - larger than $O(n!)$, although this is smaller than the $O(n!)$ of the topological ordering problem, as we have limited the number of qubits to search over
- Brute force is not possible, so
- Can we even find the largest strategy efficiently?
- If we can't, can we at least find a pretty large one?

How Hard is Strategy Maximization?

It turns out that finding the largest strategy is NP-hard! This means that there is no polynomial-time algorithm that can compute the largest strategy (unless you've just solved a millenium problem). How do we know?

How Hard is Strategy Maximization?

It turns out that finding the largest strategy is NP-hard! This means that there is no polynomial-time algorithm that can compute the largest strategy (unless you've just solved a millenium problem). How do we know?

Given the graphs $\rightarrow$ and $\hookrightarrow$ and their adjacency matrices A and R, we know that

$$\rightarrow\hookrightarrow \text{ is acyclic} \Leftrightarrow AR \text{ is nilpotent}$$

How Hard is Strategy Maximization?

It turns out that finding the largest strategy is NP-hard! This means that there is no polynomial-time algorithm that can compute the largest strategy (unless you've just solved a millenium problem). How do we know?

Given the graphs $\rightarrow$ and $\hookrightarrow$ and their adjacency matrices A and R, we know that

$$\rightarrow\hookrightarrow \text{ is acyclic} \Leftrightarrow AR \text{ is nilpotent}$$

A matrix M is nilpotent if $\exists k, M^k = 0$. Intuitively, the i, jth entry of an adjacency matrix raised to the power k gives the *number of length-k paths between nodes i and j in the graph*.

How Hard is Strategy Maximization?

It turns out that finding the largest strategy is NP-hard! This means that there is no polynomial-time algorithm that can compute the largest strategy (unless you've just solved a millenium problem). How do we know?

Given the graphs $\rightarrow$ and $\hookrightarrow$ and their adjacency matrices A and R, we know that

$$\rightarrow\hookrightarrow \text{ is acyclic} \Leftrightarrow AR \text{ is nilpotent}$$

A matrix M is nilpotent if $\exists k, M^k = 0$. Intuitively, the i, jth entry of an adjacency matrix raised to the power k gives the *number of length-k paths between nodes i and j in the graph*.

So, if an adjacency matrix is nilpotent, it cannot be cyclic because there is a maximum number of steps k between any two nodes in the graph.

How Hard is Strategy Maximization?

Now, we know that[2]

$$AR \text{ is nilpotent} \Leftrightarrow \exists P, P^T A(RP) \text{ is strictly lower triangular}$$

[2] I have had a hard time deriving this myself. Resources say that it is true due to the Cayley-Hamilton theorem.

How Hard is Strategy Maximization?

Now, we know that[2]

$$AR \text{ is nilpotent} \Leftrightarrow \exists P, P^T A(RP) \text{ is strictly lower triangular}$$

This statement of triangularity has been directly studied in another way: Wilf's problem studies the complexity of permuting the rows and columns of a matrix to make it strictly upper/lower triangular.

[2] I have had a hard time deriving this myself. Resources say that it is true due to the Cayley-Hamilton theorem.

How Hard is Strategy Maximization?

Now, we know that[2]

$$AR \text{ is nilpotent} \Leftrightarrow \exists P, P^T A(RP) \text{ is strictly lower triangular}$$

This statement of triangularity has been directly studied in another way: Wilf's problem studies the complexity of permuting the rows and columns of a matrix to make it strictly upper/lower triangular.
It can be shown that Wilf's problem reduces to the strategy maximization[3], therefore qubit recycling strategy maximization is proven to be $\boxed{\text{NP-hard}}$!

[2] I have had a hard time deriving this myself. Resources say that it is true due to the Cayley-Hamilton theorem.

[3] Not important how, TL;DR is that it involves padding a matrix until it becomes a valid QDG of some circuit

Solving NP-hard Strategy Maximization

Now that we know how hard the problem is, we begin to solve the strategy maximization problem in a reasonable amount of time using known techniques for Wilf's problem:

	0	1	2	3
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1

(a)

	0	1	2	3
3	1	0	0	1
2	1	0	1	1
1	1	1	0	1
0	1	1	1	1

(b)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(c)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(d)

Fig. 3. Demonstrating the solving procedure using the circuit in Fig. 1a

Solving NP-hard Strategy Maximization

Now that we know how hard the problem is, we begin to solve the strategy maximization problem in a reasonable amount of time using known techniques for Wilf's problem:

	0	1	2	3
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1

(a)

	0	1	2	3
3	1	0	0	1
2	1	0	1	1
1	1	1	0	1
0	1	1	1	1

(b)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(c)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(d)

Fig. 3. Demonstrating the solving procedure using the circuit in Fig. 1a

Core idea: shift zeros to the top right corner (make matrix lower triangular). The indices of the diagonal elements of the upper-right-side submatrix correspond with a highly-optimal rewriting solution!

Solving NP-hard Strategy Maximization

Now that we know how hard the problem is, we begin to solve the strategy maximization problem in a reasonable amount of time using known techniques for Wilf's problem:

	0	1	2	3
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1

(a)

	0	1	2	3
3	1	0	0	1
2	1	0	1	1
1	1	1	0	1
0	1	1	1	1

(b)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(c)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(d)

Fig. 3. Demonstrating the solving procedure using the circuit in Fig. 1a

Core idea: shift zeros to the top right corner (make matrix lower triangular). The indices of the diagonal elements of the upper-right-side submatrix correspond with a highly-optimal rewriting solution! Above, the indices are $(2, 3), (1, 2)$ - that's the same as the previously-mentioned strategy $\{1 \hookrightarrow 2, 2 \hookrightarrow 3\}$! Wow!

Recap

So far, we've seen that:

- It's too difficult to search for topologically-identical but smaller quantum circuits

Recap

So far, we've seen that:

- It's too difficult to search for topologically-identical but smaller quantum circuits
- It's much easier to search for valid *recycling strategies*

Recap

So far, we've seen that:

- It's too difficult to search for topologically-identical but smaller quantum circuits
- It's much easier to search for valid *recycling strategies*
- Finding the largest recycling strategy is NP-hard, proven via a reduction from Wilf's matrix triangularization problem

Recap

So far, we've seen that:

- It's too difficult to search for topologically-identical but smaller quantum circuits
- It's much easier to search for valid *recycling strategies*
- Finding the largest recycling strategy is NP-hard, proven via a reduction from Wilf's matrix triangularization problem
- Diagonals on the upper-right-side submatrix of a lower-triangular adjacency matrix for a given QDG correspond with highly-optimal recycling strategies

Putting it All Together

A theoretical framework is great, but what we really want is an optimizing compiler to do all of the work for us:

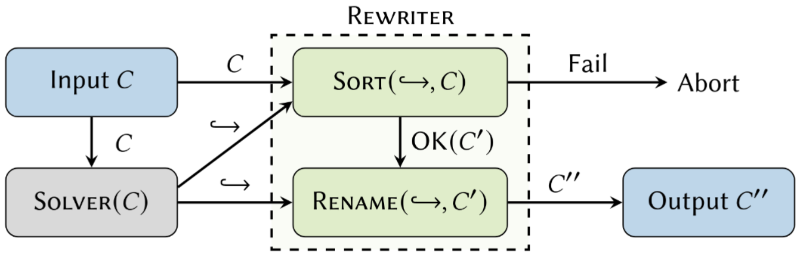


Fig. 5. Structure of the certified qubit recycler.

Putting it All Together

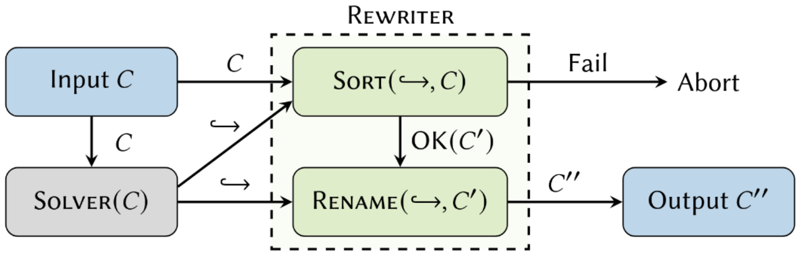


Fig. 5. Structure of the certified qubit recycler.

This compiler calls the previously-described **Solver** to generate rewriting strategies, then uses its **Rewriter** to either fail if the generated strategy is invalid, or produce a *semantically-equivalent* circuit C''.

Semantic Equivalence

- The **Rewriter** module is written in Coq, a language used for formal verification of software

Semantic Equivalence

- The **Rewriter** module is written in Coq, a language used for formal verification of software
- The **Sort** module implements the topological sort, which simultaneously sorts the circuit and detects cycles in the recycling strategy - its correctness is verified with a formal proof

Semantic Equivalence

- The **Rewriter** module is written in Coq, a language used for formal verification of software
- The **Sort** module implements the topological sort, which simultaneously sorts the circuit and detects cycles in the recycling strategy - its correctness is verified with a formal proof
- Because the topological sort is proven correct, we know that it does not alter the semantics of the original circuit C

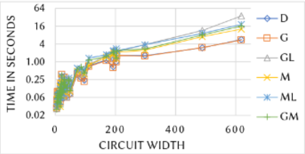
Semantic Equivalence

- The **Rewriter** module is written in Coq, a language used for formal verification of software
- The **Sort** module implements the topological sort, which simultaneously sorts the circuit and detects cycles in the recycling strategy - its correctness is verified with a formal proof
- Because the topological sort is proven correct, we know that it does not alter the semantics of the original circuit C
- The **Rename** module connects together the topologically-deformed circuit C' according to the provided strategy $\hookrightarrow$, also verified to be semantic-preserving

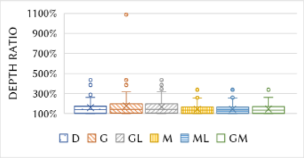
Experimental Results

Circuit	W	#Recycled qubits (the more the better) P	D	G	GL	M	ML	GM
pdc_307	619	464	505	505	505	508	508	508
spla_315	489	401	407	407	407	407	407	407
hwb9_304	170	81	121	121	119	119	119	121
ex5p_296	206	107	127	127	127	125	125	127
e64-bdd_295	195	114	126	126	126	126	126	126
hwb8_303	112	52	73	73	73	73	73	73
hwb7_302	73	31	45	45	45	44	44	45
hwb6_301	46	20	22	22	23	22	22	22

(a) For each circuit, we list its width in column "W", and the number of recycled qubits using various methods in sub-columns of "# Recycled qubits". Each sub-column corresponds to a method as follows: "P": those reported in [Paler et al. 2016]; "D": our implementation of [DeCross et al. 2023]'s algorithm; "G": Greedy; "M": Max0s; "GL": Greedy+LA; "ML": Max0s+LA; "GM": Greedy+Max0s. The best results among the methods are highlighted.



(b) Average time consumption.



(c) Box plot of the ratios of circuit depths after and before recycling.

Takeaways

Paper DOI

- *Qubit Recycling* aims to reduce the number of qubits used in a circuit
- Existing Qubit Recycling strategies both do not always provide optimal solutions and are not guaranteed to maintain semantics (behavior) of a quantum circuit
- Jiang introduces *Qubit Dependency Graphs* as a key generalization, allowing for verifiable and usually-optimal recycling solutions
- Recycler algorithm is formally verified in the Coq Proof Assistant, showing that it always maintains the semantics of rewritten circuits