Uncomputing Ancilla Qubits in Quantum Circuits

Ashutosh Tiwari, Ranjani G. Sundaram, Himanshu Gupta, C.R. Ramakrishnan, Nengkun Yu
Stony Brook University, NY

***Abstract*—Uncomputation of ancilla qubits is essential in quantum computing to reset auxiliary qubits to a zero state, ensuring their safe discarding and reducing resource overhead. Current methods for uncomputation either rely on manual procedures or use additional ancilla qubits to store intermediate computation results, which increases qubit storage requirements. Additionally, most in-place uncomputation techniques employ an all-or-nothing strategy, wherein they perform uncomputation only when all ancilla qubits in the given circuit can be uncomputed.**

In this work, we tackle the problem of uncomputing a maximum number of ancilla qubits in a quantum circuit with minimal qubit storage overhead. We propose a novel approach where partial uncomputation—reverting the last 'm' gates—enables the uncomputation of more qubits. We present three algorithms designed to uncompute the largest possible set of ancilla qubits and demonstrate their effectiveness through experiments on various randomly generated quantum circuits.

I. Introduction

Ancilla qubits (or ancillae) are often used in quantum programs to store temporary values to assist in the primary computation. Some examples of their uses include error correction schemes, implementing complex quantum gates, quantum state preparation, creating oracle functions for quantum algorithms, etc. Unlike temporary values in classical programs that can be discarded without consequences, ancilla qubits in quantum programs cannot be ignored or overwritten without unintended effects to the computation. This is because ancilla qubits are often entangled with other qubits (input qubits), and residual information in them can cause or negate interference.

One way to ensure the safe discarding of ancilla qubits is through uncomputation. Uncomputing ancillae involves reversing the operations that brought the qubits into their current states, effectively restoring them to their original states. An added advantage of uncomputation is the reusability of ancillae, leading to efficient management of quantum resources.

Prior Work. A popular approach for safe uncomputation of ancilla qubits was introduced by Bennett [1]. Bennett's construction ensures that any computation could be performed reversibly by storing intermediate states and later erasing them using the reverse sequence of operations. However, storing the intermediate states and reversing all the operations leads to high qubit storage overhead and increased circuit depth. To remedy these issues, recent works [7, 8] devised methods for automatic uncomputation of ancillae that did not require additional qubits and ensure that reverting operations (or uncomputation gates) are placed at the correct point within the circuit instead of appending them at the end. Unfortunately, the above works are restricted to cases where there is no cyclic dependency between the qubits while uncomputation is performed. Thus, they fail when even one ancilla in the input circuit is not uncomputable.

In our work, we extend the approach from [7] to uncompute a maximum number of ancilla qubits when uncomputation of all qubits is not possible. Through experiments, we show that uncomputing a subset of ancillae is beneficial and brings the final state of the circuit closer to the desired state.

Our Contributions. In this paper, we introduce the problem of automatic uncomputation of a maximum number of ancillae in a given circuit. We propose several approaches to tackle this problem.

- The first approach, called EXHAUSTIVE, iterates over all possible subsets of ancillae and tries to uncompute them. It then returns a modified circuit with the largest set of ancillae uncomputed.
- The second approach, called GREEDY-FULL, starts by including uncomputation gates for all ancillae even when there are cyclic dependencies. It then greedily removes all uncomputation gates from the ancillae involved in the highest number of cycles.
- Our third approach, called GREEDY-PARTIAL, also starts by including uncomputation gates for all ancillae. However, instead of removing all uncomputation gates from an ancilla in each iteration, it only removes the last 'm' uncomputation gates that contribute toward cyclic dependencies among the ancillae.
- Finally, we evaluate our algorithms on a wide range of randomly generated quantum circuits and observe that a significant number of ancillae can be uncomputed even when there are cyclic dependencies.

II. Background

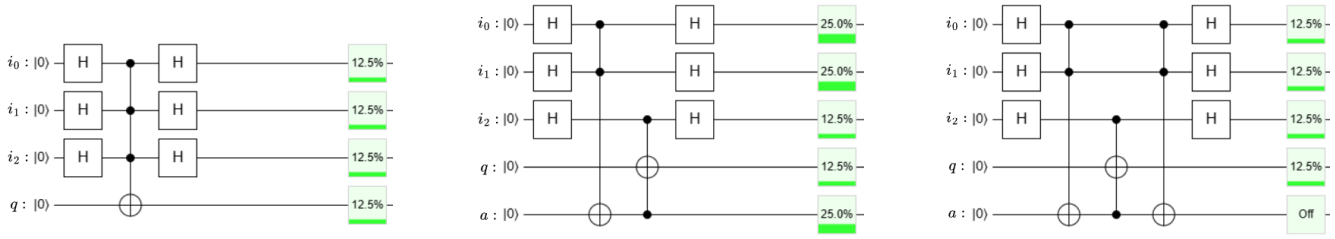
In this section, we introduce concepts fundamental to quantum programs and uncomputation.

Quantum Circuit. A quantum circuit is a model for quantum computation that describes how quantum bits (qubits) are manipulated to perform calculations. It consists of a sequence of quantum gates applied to qubits. In general, the qubits in a quantum circuit can be classified as either an input qubit or an ancillary qubit (defined below).

Ancillary Qubits. Quantum circuits often use additional "ancillary" qubits to temporarily store values and simplify calculations between the input qubits. Examples of this include the use of MULTI-CONTROL-NOT gates with ancillary qubits and Grover's algorithm.

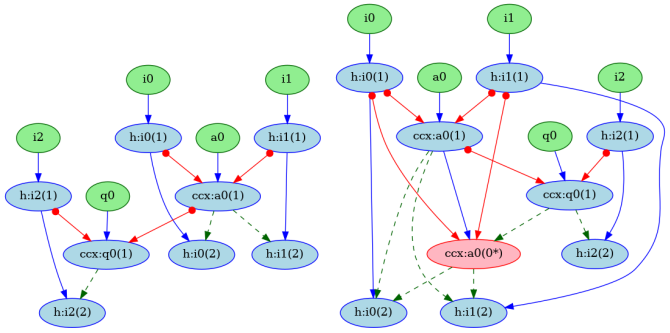
Consider the MULTI-CONTROL-NOT gate with 3 input qubits i_0, i_1, i_2 and one output qubit q, as illustrated in Figure 1(a). In this setup, the value of the qubit q will be flipped if the values of qubits i_0, i_1, i_2 are equal to 1.

This work was partly supported by the National Science Foundation awards 2106447 and 2128187.



(a) (b) (c)

Fig. 1: Representations of the MULTI-CONTROL-NOT circuit (a) without any ancilla, (b) with one ancilla that simplifies the MULTI-CONTROL-NOT gate with two CCNOT gates, (c) with one ancilla that is uncomputed. The numbers in the green boxes indicate the chance of obtaining $|1\rangle$ when the qubit is measured.



(a) (b)

Fig. 2: The circuit graph (a) for the MULTI-CONTROL-NOT circuit shown in Figure 1(b) and the corresponding uncomputation graph (b) for the circuit shown in Figure 1(c). Solid blue edges represent target edges, solid red edges indicate control edges, and dashed green edges denote anti-dependency edges. Green nodes indicate initialization nodes, blue nodes indicate the computation nodes and red nodes indicate the added uncomputation nodes.

To simplify the MULTI-CONTROL-NOT gate, we can use TOFFOLI gates (CONTROL-NOT with two controls) along with an ancillary qubit, as shown in Figure 1(b). First, a TOFFOLI gate is applied to the ancillary qubit a, with qubits i_0 and i_1 as controls. Then, a second TOFFOLI gate is applied to the output qubit q, using the ancilla a and input qubit i_2 as controls.

Uncomputation. Entanglement between ancilla qubits and input qubits may cause unwanted or negate necessary interference. For example, in the circuit shown in Figure 1(b), we observe how the entanglement between a and i_1, i_2, i_3 causes interference that results in the the probability of measuring $|1\rangle$ for i_0 and i_1 to be 0.25, while the desired probability is 0.125. One way to prevent ancillae from causing undesirable effects in the computation is through *uncomputation*.

Uncomputation in quantum circuit design utilizes the reversibility of quantum operations to clear temporary values stored in ancillary qubits, restoring each ancillary qubit to a $|0\rangle$ state. Unlike classical ancilla values, quantum ancillary qubits cannot simply be discarded after use, as this would be equivalent to measuring the ancilla. Since ancillae are often entangled with other computational qubits, such a measurement would potentially disrupt the state of these qubits, corrupting the computation. Figure 1(c) illustrates uncomputation, building on the previous MULTI-CONTROL-NOT example. Note that the probability of measuring $|1\rangle$ for the four computational qubits is 0.125 in Fig. 1(a) (no ancilla) and Fig. 1(c) (uncomputation performed).

The mathematical design for uncomputation can be generalized by the following equations from [7]. If the first m qubits of the circuit are ancillary qubits A, and the remaining qubits are in a state φ, then Equation 1 represents the mapping between the initial and final states of the circuit.

$$|0\cdots0\rangle_A \otimes \varphi \xrightarrow{||G||} \sum_{k\in\{0,1\}^m} \gamma_k |k\rangle_A \otimes \phi_k \tag{1}$$

Adding uncomputation gates to the quantum circuit ensures that the m ancillary qubits are returned to the $|0\rangle$ state at the end. The relationship between the initial and final states of the now uncomputed circuit is described by Equation 2.

$$|0\cdots0\rangle_A \otimes \varphi \xrightarrow{||\mathcal{G}||} \sum_{k\in\{0,1\}^m} \gamma_k |0\cdots0\rangle_A \otimes \phi_k \tag{2}$$

Qfree Gates. The concept of Qfree gates was first introduced in [2]. A quantum gate is considered "Qfree" if its operation on a qubit resembles the action of a function on a classical bit. For instance, the PAULI-X gate qualifies as Qfree because its operation on a qubit can be represented by a classical NOT function. The PAULI-X gate transforms the $|0\rangle$ state into the $|1\rangle$ state, and vice versa. A few more examples of Qfree gates are the CONTROL-NOT and MULTI-CONTROL-NOT gates. In contrast, the Hadamard gate is an example of a non-Qfree gate. The algorithm for automatic uncomputation in [7] requires all gates operating on ancilla qubits to be Qfree in the input circuit. As we use their approach as a subroutine, we also assume that the input circuit has only Qfree gates on the ancillae.

Circuit Graphs. The concept of circuit graphs is introduced in [7] to describe quantum circuits. A valid circuit graph for a quantum circuit is a directed acyclic graph (DAG) where the nodes represent the circuit's gates. Initialization nodes are used to denote the initialization of each qubit in the circuit, while computation nodes refer to nodes representing gates that originate from the original circuit.

There are three types of edges in this graph: control edges, target edges, and anti-dependency edges. A control edge

between two nodes $a \bullet\!\!\rightarrow b$ implies that the state of the qubit at node a acts as a control for the gate at node b. A target edge between two nodes $a \longrightarrow b$ implies that the gate described by node b acts on the state of the qubit at node a, signifying an input-output relationship. Anti-dependency edges between two nodes $a \dashrightarrow b$ show an implicit ordering of gates represented by the two nodes. These edges are constructed from the control and target edges. Any quantum circuit Q can be converted into a circuit graph G. We refer to the nodes of the circuit graph as computation nodes.

Figure 2(a) gives an example of the circuit graph constructed for the MULTI-CONTROL-NOT circuit shown in Figure 1(b). The nodes and edges are color-coded for easier identification.

Uncomputation Gates. We use the term uncomputation gates to describe quantum gates used to uncompute the state of an ancillary qubit to its previous state. These gates are inverses of the gates applied initially.

Uncomputation Graph. The uncomputation graph of a given circuit is the modified circuit graph that contains additional nodes and edges representing the uncomputation of the ancilla. If uncomputation gates are added for all ancillae, the resulting graph is a complete uncomputation graph. The nodes in the uncomputation graph corresponding to the uncomputation gates are called uncomputation nodes. These nodes are added according to the automatic uncomputation strategy described in [7]. Specifically, the computation nodes are ordered according to a topological sorting, and uncomputation nodes are added in the reverse order of the computation nodes. We describe how uncomputation nodes are added for a particular qubit. Suppose a is an ancilla we wish to uncompute. Let $a_0, a_1, \ldots, a_i$ denote the computation nodes corresponding to a in the circuit graph. Then, we first add the uncomputation node a^*_{i-1}, that reverts the state of a_i to a_{i-1}, followed by the uncomputation nodes $a^*_{i-2}, a^*_{i-3}, \ldots a^*_0$. Adding an uncomputation node a^*_j to the graph results in the following edges being added:

- A *target edge* from a^*_{j+1} to a^*_j
- For each computation node c with a *control edge* from c to a_{j+1}, either:
  - a control edge from c to a^*_j if c^* is not present in the current uncomputation graph, or
  - a control edge from c^* to a^*_j if the uncomputation node c^* is available.
- *Anti-dependency edges* from node v to a^*_j where v is any node that has a control edge from a^*_{j+1}
- *Anti-dependency edges* from a^*_j to all the nodes w where w is a node with a target edge from a node c that has a control edge to a^*_j.

Figure 2(b) shows the uncomputation graph of the MULTI-CONTROL-NOT circuit. Here, the red node $(ccx : a0(0^*))$ is an uncomputation node, while the rest are computation and initialization nodes. There are control edges (colored red) from the controls of $(ccx : a0(1))$ (that is, $(h : i0(1))$ and $(h : i1(1))$) to $(ccx : a0(0^*))$. Additionally, anti-dependency edges (dashed green) are added from $(ccx : a0(0^*))$ to $(h : i0(2))$ and $(h : i1(2))$. This is because $i0$ and $i1$ must be used as controls for the uncomputation gate CCX on a after exactly one Hadamard gate is applied on them, but before their states are changed by the second Hadamard gate.

III. Problem Formulation

In this section, we describe our motivation to attempt uncomputing a subset of ancillary qubits, when all of them are not uncomputable, and formally define our problem statement. Before this, we define some terms used in our algorithms.

Qfree Ancilla. An ancilla qubit that has only Qfree gates performed on it. In this work, we assume that all ancillae in the input circuit are Qfree.

Fully Uncomputable Ancilla. An ancillary qubit is described as fully uncomputable if all the uncomputation gates of the ancilla can be added to the quantum circuit, and bring it back to the $|0\rangle$ state.

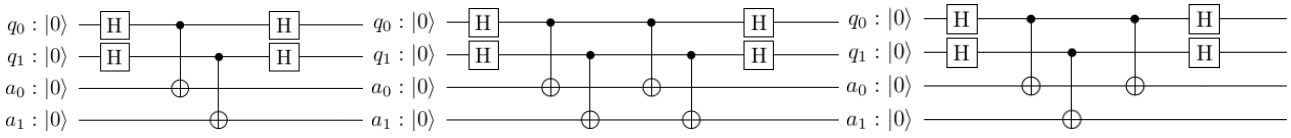
m-Partially Uncomputable Ancilla. An ancillary qubit is said to be m-partially uncomputable, if out of n total gates acting on the ancilla in the quantum circuit, only the last m gates can be uncomputed. This means that if $m = 2$, then just the final 2 gates that operate on the ancilla can have their uncomputation gates added to the quantum circuit.

Motivation for "Maximal" Uncomputation A quantum circuit is *maximally* uncomputable if only a subset of all the ancillary qubits are fully uncomputable. Similarly, a maximal uncomputation graph and the corresponding maximal uncomputation circuit are obtained by fully uncomputing a maximal set of ancillae. Prior works [2, 7] have exclusively focused on uncomputing *all* the ancillary qubits in a given circuit. However, even in a specialized setting involving only Qfree ancillae, some ancillae may not be uncomputable due to the controls being lost before the ancillae can be uncomputed. In such cases, uncomputing the other ancillae may bring the state of the input qubits closer to the desired state vector.

To illustrate the motivation behind uncomputing a subset of ancillary qubits, observe the quantum circuit shown in Figure 3(a). It consists of 2 input qubits, q_0, q_1, and 2 ancillary qubits, a_0, a_1. The state at the end of the circuit in Fig. 3(a) will be $\sum_{k\in\{0,1\}^4} \frac{1}{4} |k\rangle$, while the state of just the input qubits will be $\sum_{k\in\{0,1\}^2} \frac{1}{2} |k\rangle$.

Ideally, we would want both ancillary qubits to be uncomputed, to bring the ancillas to the $|0\rangle$ state. The ideal uncomputation circuit would look like Figure 3(b), where both the ancillas have been uncomputed, the state of the circuit is simply $|0000\rangle$ and the state of the input qubits will be $|00\rangle$.

However, let us assume that the controls to the ancillary qubit a_1 are lost, and therefore it can not be uncomputed. The only uncomputation that can be added is that for a_0 (See Fig. 3(c)). We observe that if we add the uncomputation for a_0, the state of the circuit becomes $\frac{1}{2}(|0000\rangle + |0010\rangle + |1000\rangle + |1010\rangle)$. Similarly, the state of just the input qubits becomes $\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. Thus, by uncomputing a single ancilla, we have ensured that at least one of the input qubits (q_0) when measured always gives the intended value (of the $|0\rangle$ state).



(a) A quantum circuit with 2 input and 2 ancilla qubits (b) The completely uncomputed quantum circuit (both ancillae uncomputed) (c) The quantum circuit with just one ancilla uncomputed

Fig. 3: A Quantum Circuit with complete and reduced uncomputation

With this example, we observe that if all the ancillae cannot be uncomputed, then it might be beneficial to uncompute a subset of the ancillae to obtain a maximally uncomputed quantum circuit. We validate this observation through evaluations in §VIII.

We formally define the `Maximal-Uncomputation` problem below.

`Maximal-Uncomputation` **Problem.** Given a quantum circuit C containing input qubits and Qfree ancilla qubits, construct a modified quantum circuit $\mathcal{C}$ such that a maximum number of ancilla qubits are uncomputed in $\mathcal{C}$.

IV. Related Work

Quantum Programming Languages, such as Q# [11], QWire [9], and Quipper [3] understand the importance of uncomputation, and offer methods in their programming paradigm to explicitly uncompute qubits. Designing and adding uncomputation is the responsibility of the programmer, and can cause the code to become cluttered. Other languages like ReQWire [10] and Silq [2] verify that temporary values used in a program are uncomputable, but do not synthesize uncomputation.

Automatic Uncomputation with Qubit Storage Overhead. Bennett's construction [1] is a standard method used to perform automatic uncomputation of ancillae. The key idea behind their algorithm is to store all intermediate results during computation and then undo the computation step-by-step to restore the ancillae to their original state. Some quantum programming languages, like Quipper [3] and ReVerC [6] use Bennett's construction to support automatic uncomputation. However, storing the intermediate states and reversing all the operations leads to high qubit storage overhead and increased circuit depth.

Automatic Uncomputation with No Qubit Storage Overhead. Recent works [7, 8] focus on the problem of automatic uncomputation of ancilla qubits with no qubit storage overhead. Both works operate in the restricted setting wherein all ancilla are Qfree and the uncomputation graph is acyclic. Unqomp [7] transforms the input circuit containing ancillae into a circuit where all ancillae are uncomputed. They avoid redundancies by adding uncomputation gates at the earliest possible points (without loss of controls for other gates), instead of at the end. Reqomp [8] extends Unqomp by allowing the recomputation of qubits, essentially allowing the uncomputation of ancilla before its last use, and recomputing it when needed. This way, Reqomp reduces the number of ancillae required by the computation at the cost of an increased number of gates.

In our work, we relax the assumption that the uncomputation graph must be acyclic and provide three approaches to determine the largest subset of qubits whose uncomputation does not result in cycles.

V. Exhaustive Uncomputation

In this section, we present our first algorithm called EXHAUSTIVE UNCOMPUTATION, that fully uncomputes a maximum number of ancilla qubits.

Basic Idea. The basic idea behind EXHAUSTIVE UNCOMPUTATION is to try to add the uncomputation nodes for every combination of ancillae and provide the largest subset of ancillae that can be fully uncomputed without introducing cycles in the circuit graph.

Algorithm Description. We now present the EXHAUSTIVE UNCOMPUTATION algorithm.

EXHAUSTIVE UNCOMPUTATION:

Input. Circuit graph G, ancillary qubits $\{a_1, \ldots a_n\}$.

Output. Uncomputation graph $\mathcal{G}$, where a maximum number of ancilla qubits are uncomputed.

Algorithm.

1) Let $(A^{(1)} \cdots A^{(2^n)})$ be the power set of ancillary qubits.
2) For each set $A^{(i)}$ of ancillae, create an uncomputation graph $\mathcal{G}_{A^{(i)}}$ using the UNQOMP method described in [7] by providing the circuit graph G and the list of uncomputable ancillae $A^{(i)}$
3) Among all the uncomputation graphs $\{\mathcal{G}_{A^{(i)}}\}$'s that are acyclic, pick the one that uncomputes the largest number of ancillae and return it as $\mathcal{G}$.

In the case where partial uncomputation of ancillae is not allowed, EXHAUSTIVE UNCOMPUTATION is optimal, that is, it uncomputes the largest subset of ancillae that can be uncomputed.

Algorithm Complexity. It's important to note that while this exhaustive approach guarantees finding the optimal solution, it comes at the cost of exponential time complexity $\mathcal{O}(2^a.(n + e))$, where a is the number of ancillae, n is the number of vertices and e is the number of edges present in the graph. This is due to the generation and checking of all possible subsets of the ancilla set, as well as checking for the presence of a cycle in the uncomputation graph for each subset of ancillae.

VI. Greedy-Full Uncomputation

Here, we present our second algorithm for the `Maximal-Uncomputation` problem, called GREEDY-FULL UNCOMPUTATION.

Basic Idea. In contrast to the EXHAUSTIVE UNCOMPUTATION algorithm, which added uncomputation for subsets of ancillary qubits, this approach seeks to remove cycles from the uncomputation circuit graph by greedily identifying ancillae whose uncomputation nodes can be eliminated. The key idea is to remove uncomputation nodes from the ancillae rather than add them, as this allows us to consistently determine the "worst" ancilla to remove from — specifically, the one whose uncomputation contributes the most cycles to the circuit graph. Conversely, when it comes to adding uncomputation, we cannot reliably identify the "best" ancilla since the effect of adding uncomputation on cycle count is harder to determine.

Removing Uncomputation Nodes. Before presenting our approach, we first state and prove that an uncomputation graph created by adding uncomputation nodes for uncomputable ancillae is equivalent to an uncomputation circuit graph constructed by removing uncomputation nodes that introduce cycles in the graph.

Suppose G is the circuit graph corresponding to the given quantum circuit with input qubits Q and ancillary qubits A. Let $A' \subseteq A$ be a maximal set of ancillae that can be fully uncomputed in the given circuit. Then, we have two ways of generating an uncomputation graph corresponding to the circuit that uncomputes A'. Firstly, we can create an uncomputation graph $\mathcal{G}'$ from G by adding uncomputation nodes and corresponding gates for each ancilla in A' (the resulting graph is acyclic since A' is uncomputable). Alternatively, we can start with the complete uncomputation graph $\mathcal{G}$ (all the ancillae have uncomputation gates) which may contain cycles. We can then remove the uncomputation nodes corresponding to ancillae on $A \setminus A'$ until we obtain an uncomputation graph $\mathcal{R}$ that only uncomputes A'. We prove that $\mathcal{R}$ is isomorphic to $\mathcal{G}'$.

Theorem 1: The uncomputation graph $\mathcal{G}'$ obtained by adding uncomputation nodes for ancillae in A' to G is isomorphic to the uncomputation graph $\mathcal{R}$ obtained by removing uncomputation nodes for ancillae not in A' from the complete uncomputation graph $\mathcal{G}$.

PROOF: (*Sketch*) To prove the theorem, it is sufficient to show that $\mathcal{G}'$ and $\mathcal{R}$ share the same nodes and edges. Note that G is a subset of both $\mathcal{G}'$ and $\mathcal{R}$ (as the computation nodes remain unchanged). Additionally, $\mathcal{G}'$ and $\mathcal{R}$ contain the same uncomputation nodes (the ones corresponding to ancillae in A'). Consequently, the target edges involving uncomputation nodes are also identical $\mathcal{G}'$ and $\mathcal{R}$. Thus, we only need to show that the control and anti-dependency edges involving the uncomputation nodes are the same for $\mathcal{R}$ and $\mathcal{G}'$.

1) *Control Edges.* Consider a control edge in G from an input qubit to an ancilla in A' of the form $q_i \bullet\!\!\rightarrow a_j \in G$. Note that since q is an input qubit, the control edge $q_i \bullet\!\!\rightarrow a_j^*$ is necessary for uncomputation of a. Thus, both $\mathcal{G}'$ and $\mathcal{R}$ contain the edge $q_i \bullet\!\!\rightarrow a_j^*$.
   Suppose another ancilla b instead controls the a_j, that is, there is an edge $b_i \bullet\!\!\rightarrow a_j \in G$. Then, if $b \in A'$, there must be a control edge $b_i^* \bullet\!\!\rightarrow a_j^* \in \mathcal{G}'$ (see § II). Similarly, the edge $b_i^* \bullet\!\!\rightarrow a_j^*$ is present in $\mathcal{G}$, and will not be removed from $\mathcal{R}$. Thus, we have $b_i^* \bullet\!\!\rightarrow a_j^* \in \mathcal{R}$. If b is not uncomputable, that is, $b \in A \setminus A'$, we have $b_i \bullet\!\!\rightarrow a_j^* \in \mathcal{G}'$. However, the complete uncomputation graph $\mathcal{G}$ contains the control edge $b_i^* \bullet\!\!\rightarrow a_j^*$. Thus, when b_i^* is removed to form $\mathcal{R}$, the edge $b_i^* \bullet\!\!\rightarrow a_j^*$is replaced with the edge $b_i \bullet\!\!\rightarrow a_j^*$. Thus, $\mathcal{G}'$ and $\mathcal{R}$ contain the same control edges.
2) *Anti-dependency Edges.* Since $\mathcal{G}'$ and $\mathcal{R}$ have been shown to contain the same set of control and target edges, the resulting anti-dependency edges will also be identical.

We have shown that $\mathcal{G}'$ is isomorphic to $\mathcal{R}$.

■

Algorithm Description. We now present the GREEDY-FULL UNCOMPUTATION algorithm.

GREEDY-FULL UNCOMPUTATION:

Input. Circuit graph G, ancillary qubits $\{a_1, \ldots a_n\}$.

Output. Uncomputation graph $\mathcal{G}$, where a maximal set of ancilla qubits are uncomputed.

Algorithm.

1) Let $\mathcal{G}_{init}$ be the graph obtained from G by adding uncomputation nodes and corresponding edges for every ancilla qubit.
2) While $\mathcal{G}_{init}$ contains cycles, do
   a) Let $\mathcal{S}$ be the set of all simple cycles in $\mathcal{G}_{init}$ obtained using Johnson's algorithm [5].
   b) For each ancilla qubit a,
      - Let c be the number of simple cycles that contain one or more of a's uncomputation nodes.
   c) Let a be the ancilla with maximum c.
   d) Let $\mathcal{U}$ be the reverse topological sort of all uncomputation nodes of a.
   e) For each uncomputation node a_j^* in $\mathcal{U}$,
      - Identify the node a_{j+1} in G that sets the state of a to the same state as a_j^*.
      - Add control and anti-dependency edges from a_{j+1} to the nodes previously controlled by a_{j+1}^*.
   f) Remove the uncomputation nodes (and by extension, the edges) of a from $\mathcal{G}_{init}$.

Algorithm Analysis. The GREEDY-FULL UNCOMPUTATION algorithm's runtime is dominated by Johnson's algorithm [5] for obtaining cycles. The runtime of the overall GREEDY-FULL UNCOMPUTATION algorithm can be generalized to $O(c.(n+e)(c+1))$, where n is the total number of vertices in the graph, e is the total number of edges, and c is the total number of simple cycles in the uncomputation graph. In the worst case, Johnson's algorithm will execute c times, i.e. until all the cycles are found and removed. Since the number of cycles in a general graph can be exponential in the number of vertices, we bound the number of cycles discoverable to $500,000$ in our evaluations.

VII. GREEDY-PARTIAL UNCOMPUTATION

This section presents a modified version of the GREEDY-FULL algorithm called GREEDY-PARTIAL UNCOMPUTATION that allows some ancillae to be partially uncomputed to fully uncompute a greater number of ancillae.

Basic Idea. The EXHAUSTIVE UNCOMPUTATION and GREEDY-FULL UNCOMPUTATION algorithms utilize an all-or-nothing strategy, where they either fully uncompute an ancilla or leave it entirely uncomputed. However, in certain scenarios, the partial uncomputation of some ancillae can be advantageous. While uncomputation nodes are added in the reverse topological order, the uncomputation algorithm in [7] enables the replacement of any ancilla node c used as a control, with an equivalent c^*, if c has been uncomputed already (see § II). Let us assume that adding uncomputation nodes after c^* introduces cycles. The GREEDY-PARTIAL UNCOMPUTATION algorithm deletes the uncomputation nodes descending from c^* to remove such cycles and still have controls (used for uncomputation) for other ancillae available. This ensures we maximize the number of uncomputed controls available, facilitating the full uncomputation of other ancillae.

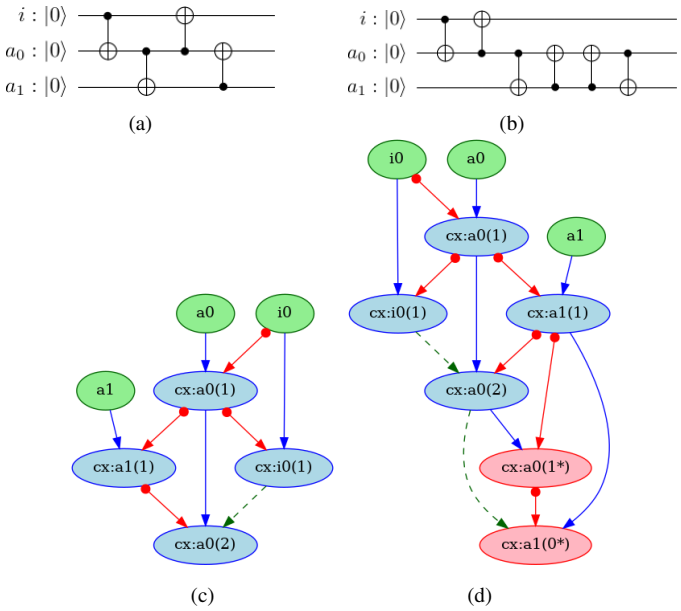


Fig. 4: An example showing how the partial uncomputation of an ancilla can enable the full uncomputation of other ancillae. (a) A quantum circuit where the controls of the ancillae can not be fully recovered. (b) The partial uncomputation of ancilla a_0 to facilitate the full uncomputation of ancilla a_1. (c) The circuit graph corresponding to (a). (d) The uncomputation graph corresponding to (b).

Example. Consider the circuit in Figure 4(a), which includes one input qubit i and two ancillae, a_0 and a_1. Its corresponding circuit graph is depicted in Figure 4(c). The controls of the ancillae are altered in such a way that they cannot be fully restored. Specifically, both the EXHAUSTIVE UNCOMPUTATION and GREEDY-FULL UNCOMPUTATION algorithms determine that none of the ancillae can be completely uncomputed without introducing cycles in the uncomputation graph. Upon closer examination, if the ancilla a_0 is reverted to its state before the second CONTROL-NOT gate targeting it, this partially uncomputed state can fully uncompute a_1. This is illustrated in the circuit graph shown in Figure 4(d), where the partial uncomputation of a_0 is represented by the node $(cx : a0(1^*))$. Here, the state of a_0 is reset to $(cx : a0(1))$, allowing this uncomputation node to act as a control that fully uncomputes a_1 without creating cycles. The equivalent circuit demonstrating partial uncomputation is presented in Figure 4(b).

Algorithm Description. We now present the GREEDY-PARTIAL UNCOMPUTATION algorithm.

GREEDY-PARTIAL UNCOMPUTATION:

Input. Circuit graph G, ancillary qubits $\{a_1, \ldots a_n\}$.

Output. Uncomputation graph $\mathcal{G}$, where a maximal set of ancilla qubits are uncomputed.

Algorithm.

1) Let $\mathcal{G}_{init}$ be the graph obtained from G by adding uncomputation nodes and corresponding edges for every ancilla qubit.
2) While $\mathcal{G}_{init}$ contains cycles, do
   a) Let $\mathcal{S}$ be the set of all simple cycles in $\mathcal{G}_{init}$ obtained using Johnson's algorithm [5].
   b) For each ancilla qubit a,
      - Let c be the number of simple cycles that contain one or more of a's uncomputation nodes.
   c) Let a be the ancilla with maximum c.
   d) Let $\mathcal{U}$ be the reverse order of all uncomputation nodes of a.
   e) For each uncomputation node a_j^* in $\mathcal{U}$,
      - If the node a_j^* does not have a control edge to an uncomputation node or is marked as **visited** (in a previous iteration of 2):
        – Identify the node a_{j+1} in G that sets the state of a to the same state as a_j^*. Add control and anti-dependency edges from a_{j+1} to the nodes previously controlled by a_{j+1}^*. Remove the uncomputation nodes (and by extension, the edges) of a_j^* from $\mathcal{G}_{init}$.
      - If the node a_j^* has at least one control edge to an uncomputation node:
        – Mark this node as **visited**. Go to step 2.

Algorithm Analysis. Like the GREEDY-FULL UNCOMPUTATION algorithm, Johnson's algorithm for obtaining cycles dominates the GREEDY-PARTIAL UNCOMPUTATION algorithm's runtime. As in GREEDY-FULL UNCOMPUTATION, we limit the number of discoverable cycles to $500,000$ during evaluations.

VIII. Evaluation

This section evaluates the three algorithms across various quantum circuits with different designs. The performance of the algorithms is evaluated by determining the average number of ancillary qubits each algorithm uncomputes.

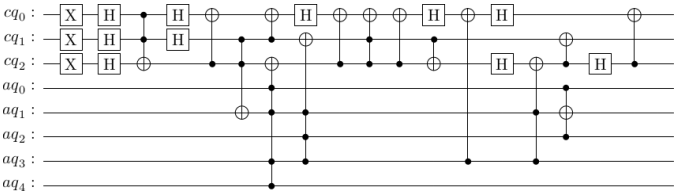


Fig. 5: A randomly-generated circuit with 3 input qubits, 5 ancillae and 15 gates (excluding the PAULI-X and HADAMARD gates)

Random Circuit Generation. We randomly generate quantum circuits using Qiskit [4]. The number of input qubits, ancillae, and gates are passed as parameters to our circuit-generating algorithm. We initialize each input qubit with a PAULI-X gate followed by a HADAMARD gate. We then randomly add CONTROL-NOT gates between the qubits, selecting the control and target qubits according to a fixed probability distribution. The resulting circuit contains approximately 80% of gates exclusively between the input qubits, 10% of gates between the input and the ancillae, and the remaining 10% between the ancillae themselves. Moreover, we randomly add HADAMARD gates on the input qubits. Figure 5 shows an example of such a randomly generated circuit. We generate a series of random circuits, varying the number of input qubits, ancillae, and gates to evaluate the performance of our algorithms.

Parameter Values. To generate circuits for evaluation, we vary one parameter at a time while keeping the other parameters fixed. We vary the following aspects of the circuit- the number of ancillary qubits (default=12), the number of input qubits (default=10), and the number of CONTROL-NOT gates (default=50). In addition to varying the number of CONTROL-NOT gates, we adjust the distribution of gate interactions between qubits. Specifically, we vary the percentage of gates acting solely on input qubits, on ancillary qubits, between input and ancillary qubits with control on the input, and between input and ancillary qubits with control on the ancillary.

Evaluation Results. The algorithms are evaluated over generated circuits as described above. Figure 6 illustrates their performance. We make the following observations:

- Generally, GREEDY-PARTIAL UNCOMPUTATION uncomputes the maximum number of ancillary qubits, outperforming the other algorithms by around 10%.
- GREEDY-FULL and EXHAUSTIVE perform relatively the same, but there are a few instances where EXHAUSTIVE uncomputes more ancillae.
- As the number of randomly placed CONTROL-NOT gates increases, fewer ancillary qubits are uncomputed.
- As the number of CONTROL-NOT gates with input qubits as controls and ancillae as targets increases, fewer ancillae are uncomputed. In contrast, if these gates are controlled by ancillae instead, we observe that more ancillae can be uncomputed. This occurs because ancillary controls are more likely to be recovered after being uncomputed themselves.

Evaluating Probability Distribution. An evaluation of the difference between the probability distributions of input qubits in the "ideal" and "maximally uncomputed" circuits is presented. These circuits are randomly generated as before with varying numbers of ancillae. The difference is quantified as the Euclidean distance between the two probability distributions. The difference between the "ideal" circuit and the "base" circuit (with no uncomputation) is calculated as a baseline, as shown in Figure 7. The results show that for

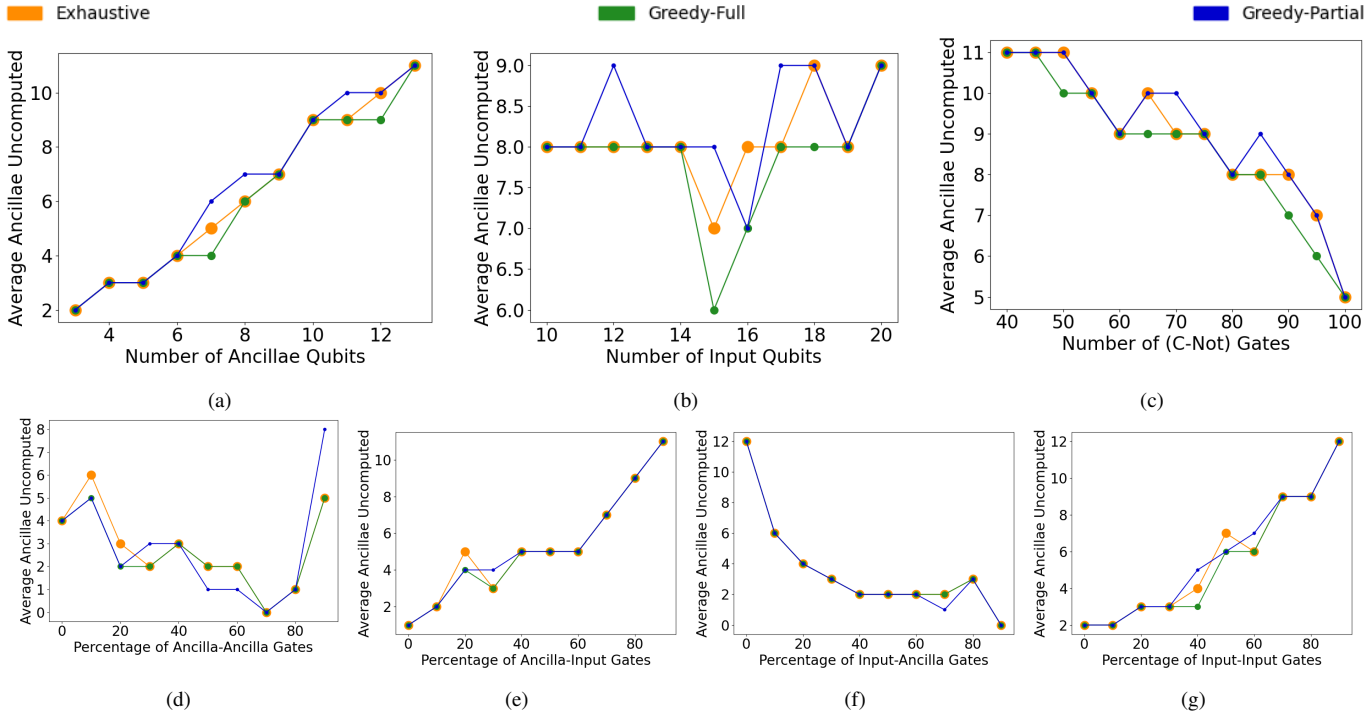


Fig. 6: Average number of ancillae uncomputed by EXHAUSTIVE, GREEDY-FULL, and GREEDY-PARTIAL algorithms for varying parameters.

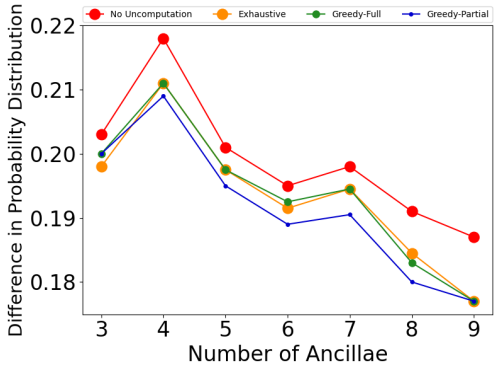


Fig. 7: Difference in probability distribution of input qubits for varying ancillae.

all methods of uncomputation, the difference in probability distribution is consistently smaller than that of the base circuit. Notably, GREEDY-PARTIAL outperforms the other methods, as it reduces the difference by uncomputing a greater number of ancillae, bringing the probability distribution of the input qubits closer to the ideal state. We note that the ideal circuit's probability distribution may not be normalized, as, in many/most cases, the ideal state with all ancillae in $|0\rangle$ state is unattainable.

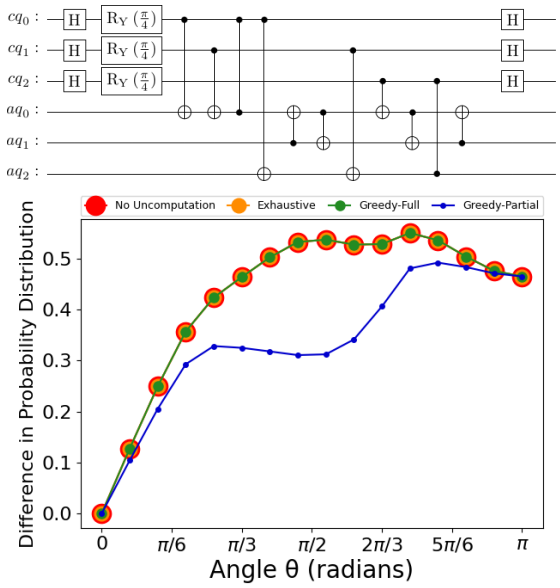


Fig. 8: Quantum Circuit (a) designed with 3 input qubits $(cq_0 - cq_2)$ and 3 ancillary qubits $(aq_0 - aq_2)$ and the corresponding difference in probability distribution betweeen the "ideal" and "maximally-uncomputed" circuit for varying input angles (b) for each uncomputation strategy.

Illustrating Benefits of Partial Uncomputation. Figures 8 and 9 demonstrate the effectiveness of GREEDY-PARTIAL UNCOMPUTATION, highlighting how different ancilla uncomputation strategies impact the final measurement probabilities of circuits shown in Figures 8 and 9. These circuits include parameterized R_Y gates on input qubits to enable arbitrary rotations, followed by QFree gates between input and ancillae.

EXHAUSTIVE and GREEDY-FULL strategies uncompute

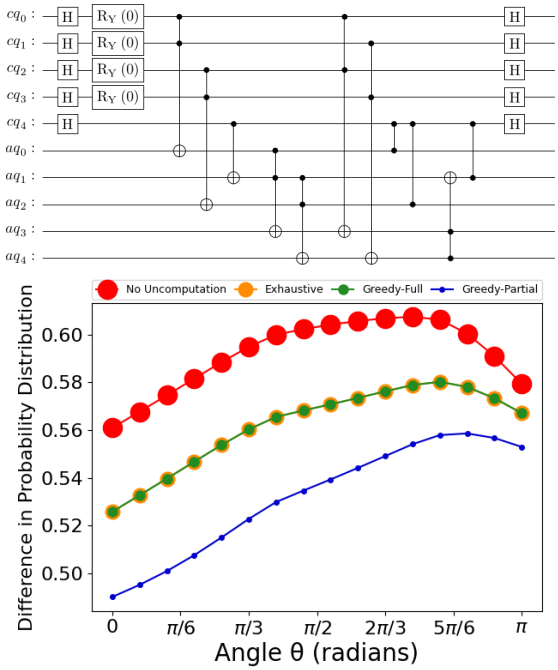


Fig. 9: Quantum Circuit (a) designed with 5 input qubits $(cq_0 - cq_4)$ and 5 ancillary qubits $(aq_0 - aq_4)$ and the corresponding difference in probability distribution betweeen the "ideal" and "maximally-uncomputed" circuit for varying input angles (b) for each uncomputation strategy.

only one ancilla ($a2$) for the circuit shown in Figure 8, while GREEDY-PARTIAL uncomputes two ancillas ($aq1$, $aq2$). Similarly, for the circuit in Figure 9 ,EXHAUSTIVE and GREEDY-FULL strategies uncompute two ancillas ($a0$, $aq2$), while GREEDY-PARTIAL uncomputes three ($aq0$, $aq2$, $aq3$).

The graphs in Figures 8 and 9 plot the difference between the measured and "ideal" probability distributions. In Figure 8, even with one uncomputed ancilla, the probability distribution for EXHAUSTIVE and GREEDY-FULL matches the initial circuit, while GREEDY-PARTIAL yields a distribution closer to the ideal. Similarly, in Figure 9, uncomputing $aq0$ and $aq2$ improves the distribution for EXHAUSTIVE and GREEDY-FULL, but the GREEDY-PARTIAL strategy achieves a much closer match to the ideal distribution.

IX. Conclusion

We address the challenge of automatically uncomputing the maximum number of ancillary qubits in a given quantum circuit. We demonstrate through experiments that uncomputing a maximal set of ancillae brings the state of the input qubits closer to the desired output state. To our knowledge, this is the first paper that considers the maximal uncomputation of a circuit in cases where not all ancillary qubits can be fully uncomputed. Additionally, we highlight the advantages of partially uncomputing ancillary qubits when full uncomputation is not feasible. Our future work consists of using quantum circuit learning to perform automatic uncomputation of ancillae to minimize gate depth overhead.

References

[1] C.H. Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 1973.

[2] B. Bichsel et al. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. ACM PLDI 2020.

[3] A.S. Green et al. Quipper: a scalable quantum programming language. ACM PLDI '13.

[4] A. Javadi-Abhari et al. Quantum computing with qiskit, 2024.

[5] D.B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 1975.

[6] A. Matthew et al. Verified compilation of space-efficient reversible circuits. In *International Conference on Computer Aided Verification*. Springer, 2017.

[7] A. Paradis et al. Unqomp: synthesizing uncomputation in quantum circuits. ACM PLDI 2021.

[8] A. Paradis et al. Reqomp: Space-constrained uncomputation for quantum circuits. *Quantum*, 2024.

[9] J. Paykin et al. Qwire: a core language for quantum circuits. ACM POPL '17.

[10] R. Rand et al. Reqwire: Reasoning about reversible quantum circuits. *Electronic Proceedings in Theoretical Computer Science*, 2019.

[11] K. Svore et al. Q#: Enabling scalable quantum computing and development with a high-level dsl. ACM RWDSL2018.