Sparse quantum state preparation with improved Toffoli cost

Felix Rupprecht[1] and Sabine Wölk[1,2]

[1]Institute of Quantum Technologies, German Aerospace Center, 89081 Ulm, Germany
[2]Center for Integrated Quantum Science and Technology (IQST), Ulm University, 89081 Ulm, Germany

The preparation of quantum states is one of the most fundamental tasks in quantum computing, and a key primitive in many quantum algorithms. Of particular interest to areas such as quantum simulation and linear-system solvers are sparse quantum states, which contain only a small number s of non-zero computational basis states compared to a generic state. In this work, we present an approach that prepares s-sparse states on n qubits, reducing the number of Toffoli gates required compared to prior art. We work in the established framework of first preparing a dense state on a $\lceil \log(s) \rceil$-qubit sub-register, and then mapping this state to the target state via an isometry, with the latter step dominating the cost of the full algorithm. The speed-up is achieved by designing an efficient algorithm for finding and implementing the isometry. The worst-case Toffoli cost of our isometry circuit, which may be viewed as a batched version of an approach by Malvetti et al., is essentially $2s$ for sufficiently large values of n, yielding roughly a $\log(s)/2$ improvement factor over the state-of-the-art. In numerical benchmarks on randomly chosen states, the cost is closer to s. With the improved isometry circuit, we examine the dense-state preparation step and present ways to optimize the joint cost of both steps, particularly in the case of target states with purely real coefficients, by outsourcing some sub-tasks from the dense-state preparation to the isometry.

1 Introduction

Preparing a quantum state $|\Theta\rangle$, i.e., finding a quantum circuit G such that $|\Theta\rangle = G|0\rangle$, is a key primitive in many areas of quantum computing, with applications ranging from linear-system solvers [1] and quantum machine learning [2] to quantum simulation [3, 4]. For this reason, different approaches for preparing generic quantum states [5–8] and states in particular formats, such as matrix-product-states [3, 4, 9–11], have been developed. While many works [6–8, 10–16] concentrate on optimizing the number of one- and two-qubit gates and/or the depth of their state-preparation circuits, the primary figure of merit for fault-tolerant quantum computation is often taken to be the number of Toffoli and/or T-gates [3, 4]. This is due to the fact that those gates are usually more costly to implement in a fault-tolerant manner compared to Clifford gates, as they are created from $|\mathrm{CCZ}\rangle$, respectively $|\mathrm{T}\rangle$ resource states built with protocols like magic-state distillation [17] and cultivation [18]. We follow this approach and decompose all circuits into the Clifford+Toffoli gate set, taking the number of qubits required, together with the number of Toffoli gates needed, as our figures of merit. As two $|\mathrm{T}\rangle$ states may be catalytically created from a $|\mathrm{CCZ}\rangle$ state and conversely a $|\mathrm{CCZ}\rangle$ state from four $|\mathrm{T}\rangle$ states [19], the Toffoli and T gate counts can be easily converted from one to another.

A class of quantum states of particular interest [1, 4] is the class of sparse quantum states. An s-sparse quantum state on n qubits is a linear combination $|\Theta\rangle = \sum_{i=0}^{s-1} c_i |C_i\rangle$ of computational basis states with $s \ll 2^n$, i.e., the number of computational basis states with non-zero amplitude is small compared to a generic state in the Hilbert space. Approaches for efficiently preparing

Felix Rupprecht: felix.rupprecht@dlr.de

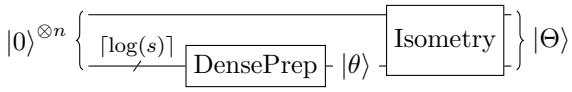


Figure 1: Sparse state-preparation circuit. First, the coefficients of the sparse state $|\Theta\rangle$ are encoded into a dense state $|\theta\rangle$ within a subspace register. Subsequently, this dense state is transformed into the target state $|\Theta\rangle$ via an isometry.

such sparse states have been studied in multiple works [4, 12–16, 20–22] and can, for the most part, be categorized into two groups. In [12, 13, 21, 22], the sparse state is built iteratively in s steps, where each step prepares a single basis state with the corresponding coefficient. The non-Clifford gate cost of each iteration step is mostly given by the implementation of a multi-controlled rotation gate or an equivalent construction. As detailed in Appendix D, the worst-case number w of control qubits for the multi-controlled gates is given by n in [21], $\lceil n/\log(n) \rceil + 1$ in [12], $\lceil \log(s) \rceil$ in [13] and the Hamming weight of the respective state in [22]. Here and in the rest of this work the logarithm is base-2. The w-controlled rotations can be implemented via a ladder of $w-1$ AND gates [23, 24] and a controlled rotation. If we allow an approximation error of ϵ for the whole preparation circuit, each controlled rotation may be implemented with Toffoli cost $\lceil \log(s/\epsilon) \rceil - 2$ within the well-known [5, 25] phase gradient framework (cf. Section 4) or alternatively via synthesized approximation [26, 27] of two uncontrolled rotations at similar cost. The overall Toffoli count of this group of approaches hence exceeds $s\lceil \log(s) \rceil$.

Instead of implementing s successive rotations, the second group of algorithms [4, 14, 16, 20], which includes the approach presented in this paper, consists of a two-step process depicted in Figure 1. In *Step 1*, a dense state $|\theta\rangle = \sum_{i=0}^{s-1} c_i |f(i)\rangle$ is prepared in a $\lceil \log(s) \rceil$-qubit subspace register, with f being a bijection from $[s]$ to an s-element subset of $[2^{\lceil \log(s) \rceil}]$. Note that by $[k]$, we denote the set of integers $\{0, \ldots, k-1\}$. Then, in the subsequent *Step 2*, an isometry is applied, mapping $|f(i)\rangle$ to $|C_i\rangle$, yielding the desired state $|\Theta\rangle$. The Toffoli cost of this approach is the cost of the dense state preparation, which is usually implemented with a Grover-Rudolph style algorithm [5], plus the cost of the isometry. The costs for the different isometry approaches in the references are investigated in Appendix D and listed in Table 1. Since in [4, 14] the bijection f is chosen to be the identity, a generic permutation of basis states must be implemented as the isometry circuit in their approaches. In [4], this incurs a Toffoli cost for the isometry of $2s(\lceil \log(s) \rceil - 1) + \tilde{s}$, where here and in the rest of the paper, we define $\tilde{s} := 2^{\lceil \log(s) \rceil}$. That said, the freedom of choosing a non-trivial bijection allows for reducing the cost of the isometry step. Indeed, as we will recall

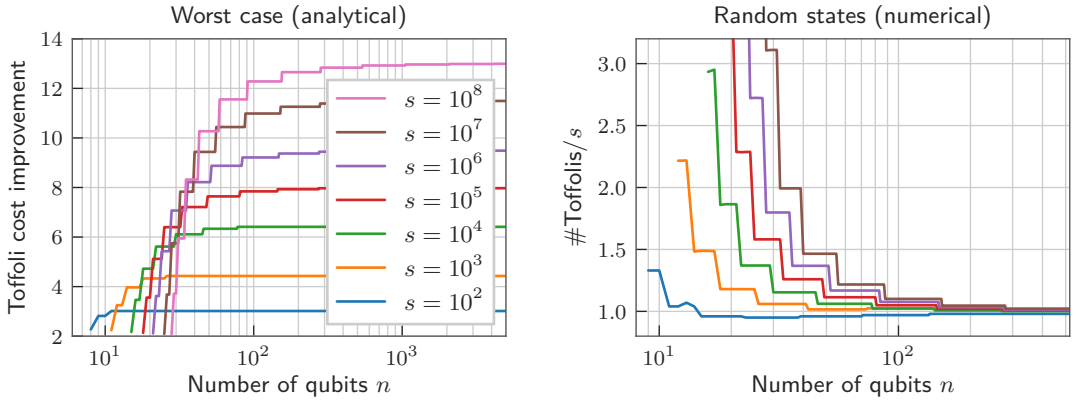


Figure 2: (Left) Improvement factor of the Toffoli cost for implementing the isometry step in this work (1) over the method of [20]. For both approaches, we considered the worst-case cost. Since the upper bound (1) is quite loose when s and m are close, the improvement according to (1) reaches a maximum at a certain n and then decreases, even though the actual improvement grows monotonically with n toward an asymptote. At each n, we therefore take the maximum improvement factor from all $n' \leq n$ in order to avoid this misrepresentation. (Right) Number of Toffolis required for implementing the isometry step with our algorithm for random s-sparse states on n qubits divided by s.

in Section 2, the method of [20] with the multi-controlled X gates implemented via AND ladders, achieves a worst-case isometry cost of $s(\lceil\log(s)\rceil - 1)$ Toffolis while requiring $\lceil\log(s)\rceil - 2$ ancilla qubits. However, with the dense state preparation in Step 1 being implementable with $2.5\tilde{s}$ Toffolis or better (cf. Section 4), even in this improved variant the isometry is still the bottleneck of the full algorithm and warrants further optimization. Since [20] gives the lowest Toffoli count of the second group of algorithms and the cost of the approaches in the first group exceeds $s\lceil\log(s)\rceil$ we take reference [20] as the baseline for the rest of the paper.

We note that there are other approaches to sparse state preparation, potentially with a smaller number of required gates [4, 7, 28]. However, the proposals require a substantial amount of additional qubits.

The contributions of this paper are as follows. We improve the Toffoli cost of sparse state preparation by constructing isometry circuits with the worst-case Toffoli count bounded by

$$\left\lceil \frac{s}{m} \right\rceil \left(2m + \frac{\log(\tilde{s}/m)}{2} - 3 \right) + \log\left(\frac{\sqrt{\tilde{s}^3}}{2\sqrt{m}} \right) \tag{1}$$

and ancilla cost $\lceil\log(s)\rceil - 1$. Here, $m := \max_{p\in\mathbb{N}}(2^p \mid 2^p \leq n - \lceil\log(s)\rceil)$ denotes the size of the largest register that fits in the n system qubits without the $\lceil\log(s)\rceil$-qubit subspace register and whose qubit number is a power of two. The Toffoli-cost improvement for the isometry circuit compared to [20] is plotted in Figure 2 on the left. The algorithm enabling these savings may be seen as a batched version of the approach in [20]. There, in the inverse circuit of the isometry, sequentially, each target basis state in $|\Theta\rangle$ is mapped from the full space into the subspace, requiring a $\lceil\log(s)\rceil$-controlled-X gate for setting the qubits outside the subspace register to zero. The main idea of this paper is to create batches of states of size m and, instead of using m individual multi-controlled-X gates for the zeroing, use a single call of a partial unary iteration circuit for doing so. The classical algorithm for finding the isometry circuit with classical runtime $O(s^2n)$ is explained in Section 2. Applying the algorithm to random s-sparse states in Figure 2 (right) shows Toffoli counts approaching s for sufficiently large n, which is roughly half the worst-case cost.

A partial unary iteration circuit can be seen as a variant of the standard unary iteration circuit $U = \sum_{i=0}^{2^u-1} |i\rangle\langle i| \otimes U_i$ proposed in [23]. It does not iterate over all basis states $[|0\rangle, |2^u\rangle]$ of a u-qubit address register, but only over an interval $[|l\rangle, |r\rangle]$. Throughout the paper, the square brackets denote intervals of consecutive basis states ordered according to their binary representation in big-endian. In Section 3, we consider two types of such partial unary iteration (PUI) circuits, which we

State	Isometry			Dense state preparation	
	Toffolis	Qubits	Reference	Toffolis	Qubits
generic	$2s(\lceil\log(s)\rceil - 1) + \tilde{s}$	$n + 5\lceil\log(s)\rceil - 3$	[4]	$2\tilde{s}/2^r + b(\lceil\log(s)\rceil - 1)(2^r - 1)$	$b2^r - r + 2\lceil\log(s)\rceil$
	$s(\lceil\log(s)\rceil - 1)$	$n + \lceil\log(s)\rceil - 2$	[20]		
	$\lceil\frac{s}{m}\rceil(2m + \log(\tilde{s}/m) - 3)$	$n + \lceil\log(s)\rceil + 1$	rPUI + sign fix		
	$3s(n-1)$	$2n - 1$	[14] *	$3\tilde{s}/2^r + (b\lceil\log(s)\rceil - b + 1)(2^r - 1)$	
	$2s(\lceil\log(2s)\rceil - 1) + n - \lfloor\log(2s)\rfloor$	$2n - \lfloor\log(2s)\rfloor$	[16] *		
	$\lceil\frac{s}{m}\rceil(2m + \frac{1}{2}\log(\tilde{s}/m) - 3) + \frac{1}{2}\log(\tilde{s}^3/4m)$	$n + \lceil\log(s)\rceil - 1$	uPUI		
real	$\lceil\frac{s}{m}\rceil(2m + \log(\tilde{s}/m) - 3)$	$n + \lceil\log(s)\rceil + 1$	rPUI + phase enc.	$\tilde{s}/2^r + b(\lceil\log(s)\rceil - 2)(2^r - 1)$	

Table 1: Upper bounds for the Toffoli and qubit costs of the isometry and dense-state-preparation steps for generic states and states with real coefficients. The rows without a reference correspond to our algorithm variants explained in Section 4: restricted PUI with integrated QROAM sign-fix, unrestricted PUI and lastly restricted PUI with integrated QROAM sign-fix and phase encoding. The counts for the entries marked with * are derived in Appendix D. We write $\tilde{s} := 2^{\lceil\log(s)\rceil}$ and let m be the greatest power of two such that $m \leq n - \lceil\log(s)\rceil$. For the first two isometry approaches from the literature, the leading Toffoli-cost term of the dense state preparation may be assumed to be 2 instead of 3, since a slight modification of the isometry circuits, analogous to our work in Section 4, allows the dense state preparation registers to be uncomputed at no extra cost. For simplicity, we choose a single r for all clean QROAM calls in the dense state preparation, where b is the number of bits for the rotation angles. We exclude the costs for preparing the phase-gradient state and do not count its qubits. Moreover, we assume that the number of qubits is largest during the QRO(A)Ms and not during the rotations.

call restricted and unrestricted, and analyze their cost. A restricted partial unary iteration circuit guarantees that U_i is non-trivial only for states $|i\rangle \in [|l\rangle, |r\rangle]$ within the target interval, while the unrestricted version, with an improved Toffoli count, relaxes this guarantee by allowing the circuit to have non-trivial contributions $|i\rangle\langle i| \otimes \tilde{U}_i$ for $i > r$. If the unrestricted version is used, one needs to keep track of those contributions so that they can be taken into account when building subsequent parts of the circuit. For the isometry circuit in Section 2 with the Toffoli bound (1), unrestricted PUIs were applied.

With the reduced cost of the isometry, the dense state preparation can now be a major contributor to the total cost of the full sparse state preparation circuit. Therefore, in Section 4, we review recent dense state preparation techniques with their Toffoli/qubit trade-offs and analyze ways of reducing the overall cost of the full sparse state preparation algorithm. By replacing the unrestricted partial iterations in the isometry with restricted ones, we can move parts of the dense state preparation to the isometry circuit. This is particularly useful in the case of real states, where it allows us to insert the complete phase information of the state in the isometry, reducing the Toffoli cost of the dense state preparation step by a factor of up to three. The resulting worst case resource counts for the algorithms are given in Table 1.

Quantum circuits for the sparse state preparation and the necessary sub-circuits are implemented within the `Qualtran` framework [29], with the computationally more expensive classical algorithms for finding the isometry outsourced to SIMD-accelerated Rust code. The full code and other assets created for this paper are available on Zenodo [30].

2 Isometry circuit

In this section, we present the classical algorithm for constructing the improved isometry circuit, together with the bijection f. We start with the classically given target state $|\Theta\rangle = \sum_{i=0}^{s-1} c_i |C_i\rangle$ and think of the computational basis states $|C_i\rangle$ as binary bitstrings in a tableau consisting of s rows and n columns, with the state $|C_i\rangle$ constituting row i. We start indexing at 0 from the left and denote by $|e_k\rangle$ the computational basis state with qubit k set to $|1\rangle$ and all other qubits set to $|0\rangle$.

We divide the bitstrings $|C_i\rangle = |C_i'\rangle^s |C_i''\rangle^r$ at position $\lceil \log(s) \rceil$, such that $|C_i'\rangle^s$ is a state within the $\lceil \log(s) \rceil$-qubit subspace register and $|C_i''\rangle^r$ a state on the remaining qubits that make up the non-subspace register. The task of the algorithm is now to find a circuit and a bijection f such that each state $|C_i\rangle = |C_i'\rangle^s |C_i''\rangle^r$ gets mapped to $|f(i)\rangle^s |0\rangle^r$, i.e., we want to map the target state into the subspace. The isometry circuit in Step 2 is then simply the inverse of the circuit found by the algorithm.

Before giving our algorithm for finding the improved isometry circuit, we briefly recall the approach of Malvetti et al. [20], with an example circuit shown in Figure 3. They iterate over all states $|C_i\rangle = |C_i'\rangle^s |C_i''\rangle^r$ and apply the following logic. If $|C_i''\rangle^r$ is the zero state, i.e., the state $|C_i\rangle$ already lies in the subspace, set $f(i) \coloneqq C_i'$. Otherwise, it has a non-zero qubit at position l, and, using CX gates controlled on this qubit, one can transform $|C_i\rangle = |C_i'\rangle^s |C_i''\rangle^r$ into $|k\rangle^s |e_l\rangle^r$ for some basis state $|k\rangle^s$ that is not yet present in the subspace register. Note that this does not affect states already in the subspace, as the control qubit in the non-subspace register

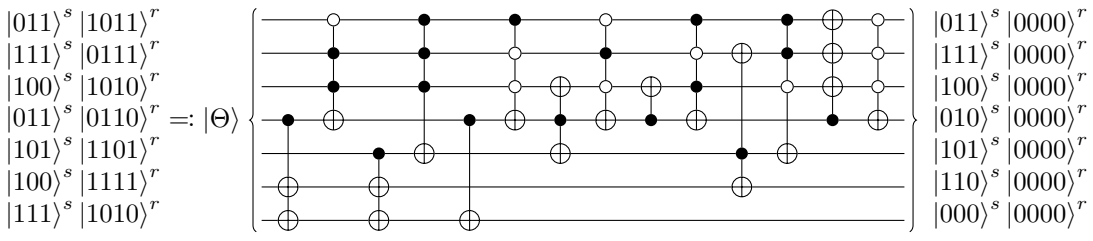


Figure 3: Isometry circuit created by the algorithm in [20], mapping $|\Theta\rangle$ to a 3-qubit subspace. For each element outside the subspace, apply CX gates to obtain $|k\rangle^s |e_j\rangle^r$ for a $|k\rangle^s$ not yet present in the subspace and a $j \in \{0, \ldots, 3\}$. Then, use a multi-controlled-X gate to transform $|k\rangle^s |e_j\rangle^r$ to $|k\rangle^s |0\rangle^r$. We index qubits from top to bottom and bitstrings from left to right.

is $|0\rangle$ for those states. The iteration step then finishes by setting $f(i) := k$ and zeroing the lone $|1\rangle$-qubit in the non-subspace register of $|k\rangle^s |e_l\rangle^r$ with a $\lceil \log(s) \rceil$-controlled-X gate controlled on the content $|k\rangle$ of the subspace register. This yields $|k\rangle^s |0\rangle^r$ as desired, while leaving states already in the subspace untouched by the choice of $|k\rangle^s$. The worst-case Toffoli cost of this approach is given by $s(\lceil \log(s) \rceil - 1)$ if the multi-controlled-X gates are decomposed into Toffoli gates with the AND-gate construction of [23, 24], requiring $\lceil \log(s) \rceil - 2$ ancilla qubits.

The main idea of this paper is to find a version of the previous algorithm that does not require s individual calls to a multi-controlled-X gate performing the zeroing $|k\rangle^s |e_l\rangle^r \mapsto |k\rangle^s |0\rangle^r$. Instead, we sequentially create $\lceil s/m \rceil$ length-m batches of the form $(|k\rangle^s |e_0\rangle^r, \ldots, |k+m-1\rangle^s |e_{m-1}\rangle^r)$ and, for each batch, use a partial unary iteration circuit introduced in Section 3 for realizing the zeroing operation

$$(|k\rangle^s |e_0\rangle^r, \ldots, |k+m-1\rangle^s |e_{m-1}\rangle^r) \mapsto (|k\rangle^s |0\rangle^r, \ldots, |k+m-1\rangle^s |0\rangle^r). \tag{2}$$

We will see in Section 3 that this partial unary iteration is most efficient if m is a power of two. Hence we take it to be the largest power of two that is less than or equal to the size of the non-subspace register, i.e., $m \leq n - \lceil \log(s) \rceil$. A formal description of the classical algorithm for finding the isometry circuit and bijection is given in Algorithm 1, with an example circuit shown in Figure 4.

If there are states not yet in the subspace, we start building a new batch by finding a state with the first qubit of the non-subspace register set to $|1\rangle$, potentially swapping qubits to do so. Using a multi-target-CX gate, this state is transformed into $|k\rangle^s |e_0\rangle^r$, where k is a runner variable that is initialized to zero at the beginning of the algorithm and incremented with each element added to a batch. The resulting state then constitutes the first element of the batch. In this way, the foremost multi-target CX in Figure 4 transforms the state $|110\rangle^s |1011\rangle^r$ into $|000\rangle^s |1000\rangle^r$. If there are still states not in the subspace or the batch, and the batch is not full, we take a state with the second qubit of the non-subspace register set to $|1\rangle$, assuming for now that such a state exists, and apply a multi-target CX controlled on this $|1\rangle$ to yield the second batch element $|k+1\rangle^s |e_1\rangle^r$. Repeating the process for the third up to the m-th qubit of the non-subspace register, or until all states are either in the batch or in the subspace, we end up with the batch $(|k\rangle^s |e_0\rangle^r, \ldots, |k+m-1\rangle^s |e_{m-1}\rangle^r)$. Then we use a partial unary iteration circuit PUI_k^{k+m-1} over the address interval $[|k\rangle, |k+m-1\rangle]$ to implement the mapping (2), zeroing out all qubits of the batch-states outside the subspace register. If we do not fill up the batch completely, m needs to be replaced by the number of elements in the batch. The circuits for the partial unary iteration are given in Section 3. They leave all states $|j\rangle^s |0\rangle^r$ for $j < k$ unchanged. However if the unrestricted version of PUI is used, states $|j\rangle^s |0\rangle^r$ with $j \geq k+m$ may be altered, and the tableau must keep track of those changes. In the algorithm, either the restricted or the unrestricted version of PUI may be used. However, unless stated otherwise, we use unrestricted PUIs because of their improved Toffoli cost. After the zeroing is done, the batch is finished, and we start building a new one.

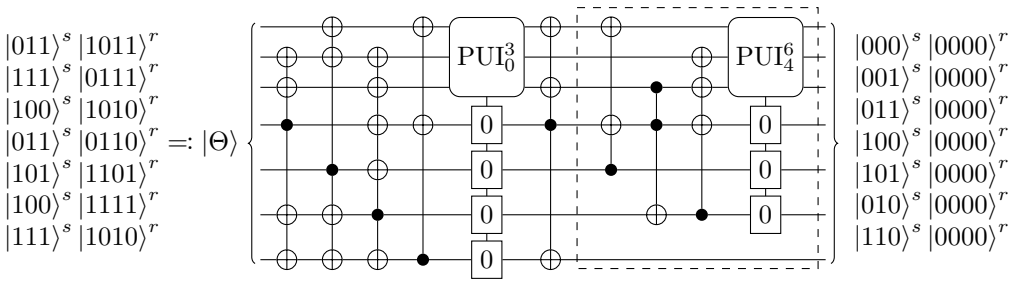


Figure 4: Our isometry circuit mapping $|\Theta\rangle$ to a 3-qubit subspace. First, we create a batch $(|0\rangle^s |e_0\rangle^r, \ldots, |3\rangle^s |e_3\rangle^r)$ using multi-controlled-CX gates and apply an unrestricted partial unary iteration PUI_0^3 with control interval $[|0\rangle^s, |3\rangle^s]$ to zero the qubits of the batch-states in the non-subspace register. Throughout the paper, we use rounded corners on boxes to emphasize that the qubits entering and exiting the box are not changed but merely serve as control qubits for the other parts of the operation. We repeat the procedure for the second batch $(|4\rangle^s |e_0\rangle^r, \ldots, |6\rangle^s |e_2\rangle^r)$. The circuits for the PUIs are given in Figure 7. Moreover, the creation of the second batch within the dotted box is analyzed in detail in Figure 5.

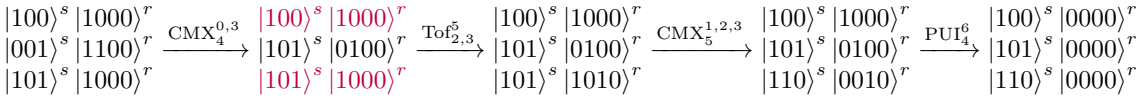


Figure 5: Creation of the second batch of the example in Figure 4 (dotted box). The first state is already in the correct form $|4\rangle^s|e_0\rangle^r$. For the second, we use a multi-target-CX gate (CMX) to bring it into batch form. Hereby, the upper indices denote target qubits, while the lower indices denote the control qubits. The last state requires a Toffoli gate controlled on the third qubit, where the purple states differ, and on the first subspace qubit, where they coincide. Subsequently, the last state is transformed so that the states constitute a full batch. The non-subspace register of the batch elements is then set to zero via an unrestricted partial unary iteration.

Looking at the example of Figure 4, four of the basis states of $|\Theta\rangle$, constituting the first batch, are mapped into the subspace and there remain three states still to be mapped. In Figure 5 we give a detailed description of the creation of the new batch, starting with the process of adding the second element. Above, we assumed that with $(l-1)$ elements in the batch there is a state with the l-th qubit of the non-subspace register set to $|1\rangle$. As can be seen in the example, this is not always the case, and we need to create such a state without altering the ones in the subspace and the batch. If there is a state with a $|1\rangle$-qubit in the non-subspace register further right than the l-th qubit, a simple swap suffices. Otherwise, we pick any $|C\rangle = |C'\rangle^s|C''\rangle^r$ not in the batch or the subspace and note that $|C''\rangle^r$ has a $|1\rangle$ within the first $(l-1)$ qubits, e.g., qubit j. By construction, there is a state $|r\rangle^s|e_j\rangle^r$ in the batch that must differ from $|C\rangle$ at some qubit u. In Figure 5 this state and $|C\rangle$ are marked in purple. If $|C\rangle$ has a $|1\rangle$ at qubit u we apply a Toffoli gate with controls u and j and the l-th non-subspace qubit as the target, creating the desired state with the l-th non-subspace qubit in $|1\rangle$. If qubit u of $|C\rangle$ is $|0\rangle$ instead, we let the Toffoli be negatively controlled on this qubit.

The bijection can be read off from the final tableau. Zeroing the elements of the final batch requires some more care. If there are no more elements outside the subspace, there are two cases. If each state within the tableau has been in a batch, we can simply apply the zeroing operation to the current batch elements as usual. However, if states lie in the subspace without having been part of a batch, we must make sure that emptying the current batch does not map those elements out of the subspace. We do so by using a restricted partial unary iteration for the zeroing operation in this case. If, during this operation, with the last element in the batch being $|r\rangle^s|e_j\rangle^r$, a non-batch state $|r\rangle^s|0\rangle^r$ is present, one ends up with the non-subspace state $|r\rangle^s|e_j\rangle^r$ after the zeroing. To fix this and map the state into the subspace, we use a single step of the Malvetti et al. algorithm with Toffoli cost $\lceil\log(s)\rceil + 1 = \log(\tilde{s}/2)$.

The Toffoli cost of the resulting circuit is the cost of the partial unary iterations plus at most

$$s - \lceil s/m\rceil + \log(\tilde{s}/2) \tag{3}$$

Toffolis. Note that the negative term is a consequence of the fact that the first element in each batch is guaranteed not to require a Toffoli. Moreover, only for the partial unary iteration circuits ancilla qubits are necessary. In numerical examples on random states (cf. Figure 2), we observe that the actual cost bounded by (3) is considerably lower, because suitable non-zero columns exist most of the time. When it comes to the multi-target CX gates in the circuit, it is worth mentioning that, using lattice surgery within a surface code, CX gates with multiple targets do not take longer than single-target CX gates [31].

For the classical cost of the algorithm, each basis state requires the tableau to be updated at most three times: once after the zeroing operation and at most twice while building the batch. Each of those updates may be performed at a cost linear in the tableau size, i.e., in $O(sn)$, and can be parallelized. Since, for all other operations, such as finding states not yet in the batch, it also suffices to scan the tableau once, and we have a constant number of such scans per basis state, the classical asymptotic cost of the algorithm is $O(s^2n)$. In Figure 10 (right), we collect some wall-clock times for our implementation [30] of the algorithm in Rust using SIMD intrinsics. On a CPU node with two AMD EPYC 7452 processors, the algorithm for $s = 10^7$ non-zero basis states runs in about 10-20 minutes. Making profitable use of multi-threading for the algorithm is non-trivial, because each individual parallelizable row operation is very cheap. We expect there to be room for improvement in the run-time of the classical algorithm, potentially by using GPUs.

3 Partial unary iteration

In this section, we construct the partial unary iteration circuits (PUI) used in Section 2 and analyze their cost. Given an interval of states $[|l\rangle^a, |r\rangle^a]$ within an w-qubit address register, the goal is to apply unitaries $U_l, \ldots, U_r$ on a target register controlled on the values in the address register, i.e., we want to find a circuit U implementing

$$U(|i\rangle^a \otimes |t\rangle^t) = |i\rangle^a \otimes (U_i |t\rangle^t) \tag{4}$$

for all $i \in \{l, \ldots, r\}$.

If $l = 0$ and $r = 2^w - 1$, this task is solved by standard unary iteration circuits based on the traversal of balanced binary trees [23] or, alternatively, skew trees [32]. Both approaches are nicely depicted in [32]. The following discussion is based on the original unary iteration circuits [23] that utilize balanced binary trees. The Toffoli cost of this approach is $2^w - 2$ (or $2^w - 1$ if controlled) and requires $w - 1$ ancillas. An example of such a full unary iteration circuit for $w = 3$ is given in Figure 6, where we chose $U_0, \ldots, U_6$ to be X gates on different qubits of the target register and U_7 to be the identity.

The core primitive of unary iteration, corresponding to the splitting of a node of the tree into its two children, may be seen in the dashed boxes within the figure. For now, we concentrate on the left box with the black dashes. Entering the box are the address qubit a_2 and an ancilla qubit d_0 created earlier (Step 0) by the AND gate (cf. Appendix A) on the first two address qubits a_0 and a_1. The ancilla d_0 is in the state $|1\rangle^{d_0}$ if those two address qubits are in the state $|1\rangle^{a_0} |0\rangle^{a_1}$. With our big-endian convention, this is the case if the 3-qubit state in the address register lies within the interval $[|4\rangle^a, |7\rangle^a]$. In general, the incoming ancilla qubit 'selects' an interval of address register values that corresponds to a node in the binary tree. Within the box, by applying an AND gate (Step 1) controlled on the ancilla d_0 and negatively controlled on the incoming address qubit a_2, the ancilla qubit d_1 created by this new AND is $|1\rangle^{d_1}$ if the address value lies in the left half of the interval, i.e., in our case if the address register is in the state $|4\rangle^a$. This left interval corresponds to one child of the node. In our case, it is a leaf node, and we can apply U_4 (Step 2) controlled on the ancilla qubit d_1. In the general case, if the half-interval at that point contains more than a single element, one recursively applies the procedure we just described until the interval has length one. When doing so, one would take the current ancilla d_1 and the next address qubit, i.e., a_3 if it existed, as the incoming qubits of the next box. In order for the ancilla d_1 in the current box to indicate that the address value lies in the right half-interval, i.e., in our case to be $|5\rangle^a$, it suffices (Step 3) to toggle it controlled on the prior ancilla d_0. This half-interval corresponds to the other child node, and since it is a leaf node we can apply U_5 controlled on the ancilla (Step 4) and uncompute the AND gate (Step 5). Note that the adjoint AND may be realized at zero Toffoli cost using measurement-based uncomputation.

If one now wants to restrict the iteration to a target interval of values $[|l\rangle, |r\rangle]$, one can simply

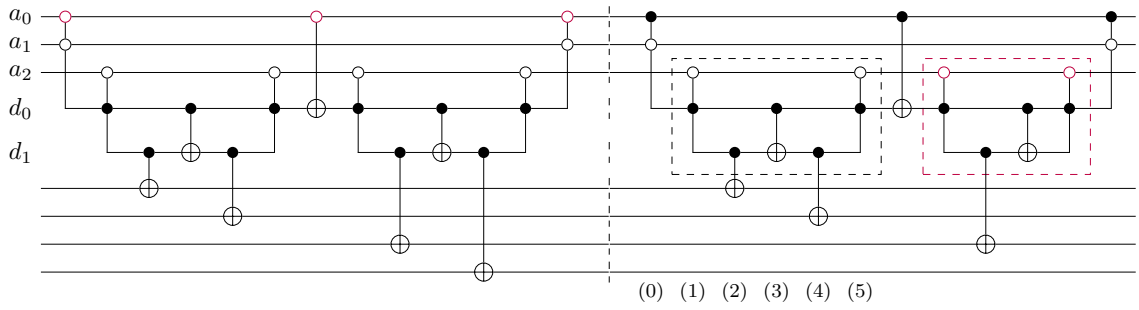


Figure 6: Full unary iteration circuit [23] applying X gates to different qubits of the lower 4-qubit target register, controlled on the values of the address register. The left and right parts of the circuit can be seen as two restricted partial unary iteration circuits on the values $[|0\rangle^a, |3\rangle^a]$ and $[|4\rangle^a, |6\rangle^a]$, respectively. In order to obtain the unrestricted versions of those PUIs shown in Figure 7, one removes the purple control dots. The purple dashed box is then simplified to the box shown in Figure 7. The definition of the AND gates is recalled in Appendix A.

limit the traversal of the balanced binary tree to values within that interval, i.e., at each step we continue the recursion on the left/right half of the current interval only if the respective half-interval overlaps with the target interval. Applying $U_l, \ldots, U_r$ when reaching the corresponding leaf nodes then implements a unitary satisfying equation (4). We choose to call the resulting circuit *restricted partial unary iteration*. Examples for such restricted PUIs are the left and right parts of the circuit in Figure 6.

In Section 2, multiple partial unary iteration circuits over successive intervals are used, with the target operations being X gates applied to individual qubits of the target register. In situations like this, it is possible to reduce the Toffoli count of the iterations by allowing the unitary

$$U = \sum_{i=l}^{r} |i\rangle\langle i|^a \otimes U_i + \sum_{i \in L} |i\rangle\langle i|^a \otimes U_{g(i)}$$

to have a non-trivial effect on address-register values that are greater than the upper limit r of the interval we iterate over. Here, $L \subseteq \{r+1, \ldots, 2^w - 1\}$ are the values of the address-register elements greater than r for which U does not act trivially on the target register. The unitary $U_{g(i)}$ applied for those values is determined by the map $g\colon L \to \{l, \ldots, r\}$. In the following, we construct a circuit PUI_l^r implementing such a unitary and call it *unrestricted partial unary iteration*.

Even though basis states $|j\rangle^a |v\rangle^t$ with $j > r$ can change, states with $j < l$ are left untouched. The application of the unrestricted form of partial unary iteration makes sense in situations where one either knows that the state to which it is applied does not have overlap with address-register components greater than the interval, or when one iterates over the full address-register values from left to right and can undo unwanted effects on a value once it lies within an active interval. For the latter, one must be able to efficiently keep track of these unwanted effects. This is the case in our isometry circuits, where they are simple bit-flips in the tableau of the classical algorithm. The two unrestricted PUI-circuits used in the isometry of Figure 4 are plotted in Figure 7.

A formal description of the classical algorithm for creating the unrestricted partial unary iteration circuit implementing U and determining the map g is given in Algorithm 2. It works similarly to the restricted version described above by recursively traversing parts of the same binary tree. However, we change the branching logic at a node depending on the overlaps of the target interval with the two half-intervals.

If both the left and right half-intervals intersect with the target interval, we use the same branching primitive as explained above. This is the case in the black dotted box on the right of Figure 7 where the left and right half-intervals are the one element sets consisting of $|4\rangle^a$ and $|5\rangle^a$, respectively, both lying within the target interval $[|4\rangle^a, |6\rangle^a]$.

In case only the right half-interval intersects with the target interval, we apply the box primitive, but only continue the recursion on the right half-interval. In the example on the right of Figure 7 only the right half-interval $[|4\rangle^a, |7\rangle^a]$ defined by the address qubit a_0 being in the state $|1\rangle^{a_0}$ has

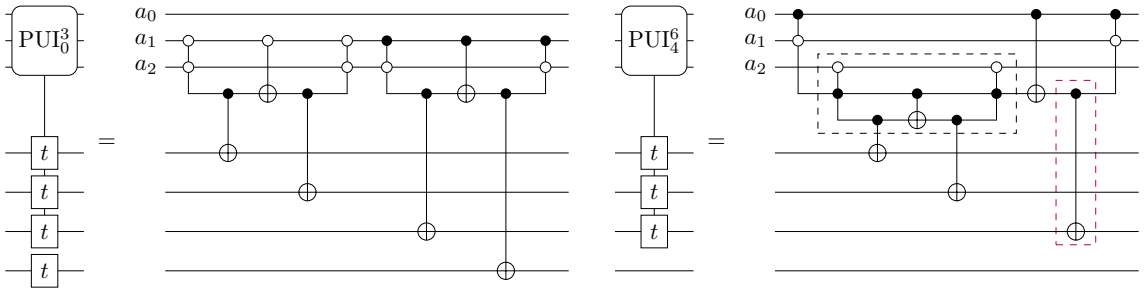


Figure 7: The two unrestricted partial unary iteration circuits on the intervals $[|0\rangle^a, |3\rangle^a]$ and $[|4\rangle^a, |6\rangle^a]$ used in Figure 4. They can be obtained from the restricted versions on the left and right-hand sides of Figure 6 via simplifications after removing the purple controls. These removals do not change how the target-X unitaries are applied on address-register values in the current interval, leaving states with address values left of the interval untouched. However, the first target-X gate in PUI_0^3 is applied not only for the address value $|0\rangle^a$ but also for $|4\rangle^a$, i.e., a value outside the interval of the PUI, where the most-significant bit changed from 0 to 1. The same is true for all other target-X gates in the first batch.

overlap with the target interval $[|4\rangle^a, |6\rangle^a]$ and consequently, all operations are controlled on $|1\rangle^{a_0}$. This assures that the partial unary iteration acts trivially on address-register content that is smaller than the target interval.

While the two cases above are the same as for restricted PUIs, the approach differs if the left half-interval intersects with the target interval, but the right one does not. In this case, we do not employ the box primitive and instead continue the recursion on the left half-interval, i.e., we ignore the address qubit that would go into the box. In Figure 7 on the left, this situation appears for a_0 corresponding to $[|0\rangle^a, |7\rangle^a]$ whose right half-interval does not intersect the target interval $[|0\rangle^a, |3\rangle^a]$, and, a second time on the right, for $[|6\rangle^a, |7\rangle^a]$ where the right half-interval consisting of $|7\rangle^a$ lies outside of $[|4\rangle^a, |6\rangle^a]$. The change in the circuit compared to the restricted PUI amounts to the removal of the purple controls on the address qubit from Figures 6. Doing so results in the target unitary not only being applied for states where the address qubit is $|0\rangle$, but also for all states where this qubit is $|1\rangle$. This means that we apply target unitaries at address states greater than the target interval.

Once a leaf, corresponding to an address-state $|j\rangle^a$, is reached, the elements we need to add to L can be inferred from j by going through all nodes in the path to the leaf where we ignored the address qubit and taking all combinations of 0 and 1 at those address positions such that at least one of them is 1. All those elements then get mapped to j under g.

It remains to determine the resources required for implementing the restricted and unrestricted partial unary iteration circuits. The number of ancilla qubits is at most $w-1$ in both cases. When it comes to the Toffoli gate count, we do not count the gates required for a single partial unary iteration circuit, but concentrate on the situation in the isometry circuits instead. We assume to successively iterate over intervals of size $m := 2^l$ for some $l \leq w$, starting from $|0\rangle^a$. We restrict to this case because it is more efficient to move in intervals whose size is a power of two than to use arbitrary interval lengths. The latter requires traversing the same tree node twice in certain situations, leading to increased Toffoli cost. As the circuits take a very simple form, the situation is also more easily analyzed than the case of general restricted and unrestricted partial unary iterations.

We start with the case where the successive partial unary iterations cover all possible address-register values, i.e., there are 2^{w-l} intervals. Each interval may be understood as selecting some value on the first $w-l$ address qubits and then doing a full controlled unary iteration on the remaining l qubits, the latter requiring $m-1$ Toffolis (resp. $m-2$ in the uncontrolled case) per interval. In the case of the restricted partial unary iterations, selecting a value on the first $w-l$ qubits requires $w-l-1$ ANDs, each with a cost of a single Toffoli. In total, one thus has a Toffoli cost for the restricted partial unary iteration of

$$\frac{S}{m}\left(m + \log(S/m) - 2\right)$$

with $S := 2^w$.

For the unrestricted case, notice that Toffolis are required for selecting a state on the first $w-l$ qubits if and only if there are at least two $|1\rangle$-qubits among them. For such a state with q qubits in state $|1\rangle$, the number of required Toffolis is $q-1$. There are $2^{w-l}(w-l)/2$ occurrences of $|1\rangle$-qubits within the first $w-l$ qubits in all possible states, $w-l$ of which belong to states with exactly one $|1\rangle$. Hence, there are a total of $2^{w-l-1}(w-l) - (w-l)$ relevant occurrences of $|1\rangle$ within $2^{w-l} - (w-l) - 1$ states, yielding

$$2^{w-l}\left(\frac{w-l}{2} - 1\right) + 1 \tag{5}$$

Toffolis for selecting on the first $w-l$ address qubits. This results in a total cost for the unrestricted case of

$$\frac{S}{m}\left(m + \frac{\log(S/m)}{2} - 2\right)$$

Toffolis.

So far, we assumed that the successive intervals cover all of the $S = 2^w$ values of the address register. However, in the isometry circuit of Section 2, it is only iterated over $s \leq S$ elements,

which need not be a power of two. In this case, assuming $2^{w-1} < s < 2^w = S$, a simple inductive argument in Appendix B shows that the number of Toffolis required for selecting on the first $w - l$ qubits in the unrestricted partial unary iteration may be upper bounded by

$$\left\lceil \frac{s}{m} \right\rceil \left(\frac{w-l}{2} - 1 \right) + 1. \tag{6}$$

The argument is based on the observation that selecting intervals closer to s contributes more Toffolis than for intervals closer to zero. For the full unrestricted partial unary iteration, one thus gets the bound

$$\left\lceil \frac{s}{m} \right\rceil \left(m + \frac{\log(S/m)}{2} - 2 \right). \tag{7}$$

Recalling that in the isometry setup $w = \lceil \log(s) \rceil$ and therefore $S = \tilde{s}$, equation (3) and the fact that the last individual PUI may be required to be restricted, leads to the bound (1). The change in Toffoli cost in case the last PUI needs to be restricted is upper bounded by $\log(S/m)$. For the restricted case, it is easy to see that the total PUI cost is bounded by

$$\left\lceil \frac{s}{m} \right\rceil (m + \log(S/m) - 2). \tag{8}$$

We note that for small values of s/m the bounds can be rather loose. As a result, considering the bounds as functions of m, there is a value at which the upper bound is minimal, after which it becomes larger again. This is in contrast to the actual Toffoli count, which is monotonically decreasing.

4 Optimizing the full circuit

So far, we have only looked at how to best implement the isometry step in the sparse state preparation scheme of Figure 1. Combining equations (3) and (7), taking into account that the last PUI needs to be restricted which adds $\frac{1}{2}\log(\tilde{s}/m)$ Toffolis compared to the unrestricted version, the cost of the isometry is

$$\left\lceil \frac{s}{m} \right\rceil \left(m + \frac{\log(\tilde{s}/m)}{2} - 2 \right) + \frac{\log(\tilde{s}/m)}{2} + s - \left\lceil \frac{s}{m} \right\rceil + \log(\tilde{s}/2)$$
$$\leq \left\lceil \frac{s}{m} \right\rceil \left(2m + \frac{\log(\tilde{s}/m)}{2} - 3 \right) + \log\left(\frac{\sqrt{\tilde{s}^3}}{2\sqrt{m}} \right),$$

where m is the largest power of two less than or equal to the number of qubits left from the n-qubit main register after the dense state is prepared, i.e., $m \leq n - \lceil \log(s) \rceil$. This is a sizeable reduction compared to the worst-case Toffoli cost of $s(\lceil \log(s) \rceil - 1)$ in [20]. We recall that m is chosen as a power of two to ensure that in each PUI a full subtree is explored. Otherwise some nodes of the subtree could be visited twice in consecutive intervals, and possibly require Toffolis in both cases.

For implementing the isometry, $\lceil \log(s) \rceil - 1$ ancilla qubits are needed. These ancillas can obviously also be used for the dense state preparation, which means that $n + \lceil \log(s) \rceil - 1$ qubits are available for this step without increasing the total qubit count of the overall sparse state preparation routine. More globally, state preparation is usually a subroutine at the beginning of a larger quantum circuit, and the number of available qubits is determined by the later parts of the circuit. In this section, we will therefore consider how to best implement the combined sparse state preparation steps with varying numbers of ancilla qubits.

We start by briefly explaining the Grover-Rudolph strategy for dense state preparation, as depicted in Figure 8 in its recent form [3, 5], where the rotations are implemented with the help of a phase-gradient state $|\psi\rangle$. Given the dense state $|\theta\rangle = \sum_{i=0}^{2^l-1} c_i |f(i)\rangle$ on $l \coloneqq \lceil \log(s) \rceil$ qubits, a two-step process is applied, the first of which prepares the correct amplitudes $|\theta'\rangle = \sum_{i=0}^{2^l-1} |c_i| |f(i)\rangle$ by a series of $l - 1$ sequential R_y-rotations, followed by the second step (dotted box in Figure 8) that uses an R_z-rotation to encode the correct phases of $|\theta\rangle$. Here, the i-th phase-gradient rotation

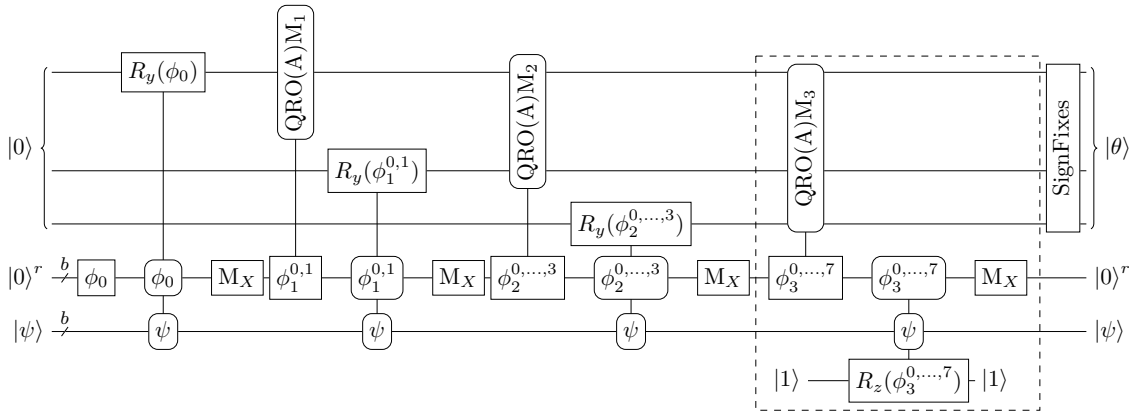


Figure 8: Grover–Rudolph type dense-state preparation circuit on three system qubits, inspired by [3, 5]. The rotation angles $\phi_0, \phi_1^{0,1}, \dots, \phi_3^{0,\dots,7}$ are loaded into an angle register that has the same size b as the register that holds the phase-gradient state $|\psi\rangle$. The states in those two registers are then used as resource states for applying R_y and R_z rotations via the phase-gradient method [5, 25]. The first angle ϕ_0 is encoded using X gates. To load the angles that depend on the contents of the previously rotated main-register qubits, QRO(A)Ms are used. After a rotation, we measure the angle-register in the X-basis (M$_X$), which yields a clean $|0\rangle^r$ and introduces ± 1 phases on the basis states of the main register. These phases can be inferred from the measurement results. While the first three rotations are required to set the correct amplitudes of the target state, the last rotation in the dashed box prepares the correct phases. At the end, we fix the signs. As before, we round the corners of gates whose qubits are not changed by the gate.

together with the preceding QRO(A)M$_i$ [5, 23, 33] may be seen as a family of controlled rotations with angles $(\phi_i^j)_{j\in[2^i]}$ where the rotation angle ϕ_i^j gets applied if the state in the register consisting of qubits $0, \dots, i-1$ is $|j\rangle$.

The rotations are done via the well-known phase-gradient method [5, 25], taking the b-qubit phase-gradient state $|\psi\rangle$ and a rotation angle ϕ stored in a b-qubit angle-register $|\phi\rangle^r$ to implement the mapping $|\psi\rangle\langle\psi| \otimes |\phi\rangle\langle\phi|^r \otimes R(\phi)$ via controlled addition/subtraction of the states in the angle- and phase-gradient registers. For the i-th rotation, in order to apply different rotation angles $(\phi_i^j)_{j\in[2^i]}$ depending on the state $|j\rangle$ of the qubits $0, \dots, i-1$, one uses some form of QRO(A)M $\sum_{j\in[2^i]} |\phi_i^j\rangle\langle 0|^r \otimes |j\rangle\langle j|$ to load $|\phi_i^j\rangle^r$ into the angle-register if the address state is $|j\rangle$. By linearity, the following phase-gradient rotation then causes the desired rotation $\sum_{j\in[2^i]} |j\rangle\langle j| \otimes R(\phi_i^j)$. The register size b is logarithmic in the accuracy of the stored rotation angles and, in most practical examples, is on the order of 20. The phase-gradient rotation itself requires b Toffoli gates, which is negligible compared to the dominating QRO(A)Ms and is thus neglected in this work. It suffices to prepare the phase-gradient state $|\psi\rangle$ once, as it can be reused. The cost for preparing $|\psi\rangle$ is therefore ignored in all our resource estimates, and we also exclude the phase-gradient register from our qubit counts. We refer to [5, 25] for the definition of the phase-gradient state $|\psi\rangle$ and for how to compute the rotation angles.

Instead of completely uncomputing the angle-register after each rotation as done in [5] we follow [3] and measure it in the X-basis. After this measurement M_X the angle register can be reset to $|0\rangle^r$. The measurement induces ± 1 phases on the main-register basis states which may be inferred from the measurement results and the angle-register content prior to the measurement. If QROAM is used for angle-loading, the junk qubits of the QROAM are treated analogously. After classically tracing the signs through the circuit, at the end we can apply sign fixes to the l-qubit target basis-states $|0\rangle, \dots, |2^l-1\rangle$. In [33], it is shown how to do these sign corrections via a QROM-lookup table on $l-1$ qubits or, if additional clean ancillas are available, via a construction similar to QROAM. The latter has Toffoli cost $2^{l-r}+2^r$ at the cost of $2^r+(l-r)$ clean ancillas for any $r < l$. We will see that a variation of our isometry algorithm, where, among other small changes, the unrestricted partial unary iterations are replaced by restricted ones, instead allows these sign-fixes to be done within the isometry at no extra cost besides the slightly more expensive restricted partial unary iterations.

Before explaining this in more detail, we review methods for loading the rotation angles with

different Toffoli–qubit trade-offs, the simplest of which is the original QROM proposal of [23]. For an address register of size k, it requires $2^k - 2$ Toffolis and $k - 1$ clean ancillas, which, for the circuit type in Figure 8, incurs a total cost of

$$\sum_{k=2}^{l}(2^k - 2) = 2^{l+1} - 2l - 2 < 2\tilde{s}$$

Toffolis with $l - 1$ ancilla qubits. If the final sign-fix is done with a lookup table of size $l - 1$, another $2^{l-1} - 2$ Toffolis are needed, leading to a total count bounded by $2.5\tilde{s}$.

If one has more ancilla qubits at hand, the clean QROAM version of [33] may be used to bring down the Toffoli cost. Combining QROM with controlled swaps, it allows loading b bits of data from a k-qubit address register using $2^{k-r} + b(2^r - 1)$ Toffolis at the cost of $b(2^r - 1) + (k - r)$ ancillas for some $r < k$. The Toffoli count is minimized for $2^r \approx \sqrt{2^k/b}$, with the cost being roughly $2\sqrt{b2^k}$, which means that with enough ancillas one can obtain almost a quadratic improvement in the number of Toffolis over the original QROM. Assuming for simplicity that we chose the same r for all angle-lookups, the overall Toffoli cost sums up to

$$\sum_{k=2}^{l}(2^{k-r} + b(2^r - 1)) = b(l-1)(2^r - 1) + (2^l - 2)2^{1-r} < 2\tilde{s}/2^r + b(\lceil \log(s) \rceil - 1)(2^r - 1)$$

respectively $3\tilde{s}/2^r + (b\lceil \log(s) \rceil - b + 1)(2^r - 1)$ if the sign-fix at the end is done with the scheme of [33]. The ancilla cost in both cases is $b(2^r - 1) + (\lceil \log(s) \rceil - r)$. Note that if, at each k, the value $2^r \approx \sqrt{2^k/b}$ is chosen optimally, one ends up with a leading-factor Toffoli cost of $2(1 + \sqrt{2})\sqrt{2b\tilde{s}}$ without the the sign-fix. Hence, it is possible to improve upon the dense state preparation with the original QROM variant almost quadratically. Finally, note that $b(2^r - 1)$ ancillas can be dirty [5] if one can tolerate worse prefactors in the Toffoli cost.

There is another variant [3] for loading the rotation angles for dense state preparation with an even better Toffoli cost using multiple angle-registers and controlled swaps. Since the minimal number of ancilla qubits is not as controllable as in the clean QROAM approach above, we, however, decide not to analyse it further here.

Comparing the Toffoli costs of the isometry and dense-state-preparation steps, in case QROM is used for the latter, the costs are distributed relatively evenly, with the upper bound (1) for the isometry being roughly $2s$ and the dense-state-preparation requiring $2.5\,\tilde{s}$ Toffolis. As seen in Figure 2, however, in practical examples the isometry cost is often close to s, making the QROM-based dense preparation step the dominant one. As noted in Section 2 this factor of two difference between the worst case and practical examples is due to the fact that usually there are enough suitable states for building a batch such that most of the time no Toffoli is required to create such a state. The distribution of the Toffoli cost among the different parts of the sparse-state-preparation stack for this QROM-based variant can be seen on the leftmost bar in Figure 9. Replacing QROM by QROAM with maximal r-value r_{max} changes the picture, and as r_{max} increases, the dense-state-preparation cost becomes increasingly negligible compared to the isometry cost. Up to a certain r_{max}, which is mostly determined by n, the extra qubits required for QROAM do not significantly exceed the number of qubits needed for the isometry, and hence the Toffoli cost can be reduced without increasing the total qubit count. As can be seen further to the right, beyond a certain point, trading additional ancilla qubits for fewer Toffolis in the dense-state-preparation gadget does not significantly improve the overall cost of sparse-state preparation.

Even though n is often large enough that QROAM with sufficiently large r values can be applied without excessive qubit overhead, it is sensible, especially when few qubits are available, to try to reduce the leading factors of the dense-state-preparation cost by other means. We have seen that applying the sign fixes for uncomputation increases the leading prefactor of the Toffoli cost from 2 to 3 (or 2.5 if standard QROM is used). In Figure 9, this part of the cost is drawn in purple. As described above, the primitive responsible for the cost is the sign-fix applied to the basis states according to the results of the X-basis measurements. Since, in the worst-case cost analysis of our isometry circuit, we assume iterating over all basis states, it is natural to consider performing the sign fix-up on the fly within the PUIs during the isometry circuit by simply applying a Z gate to a $|1\rangle$-ancilla for basis states that require a phase fix. In order to do so, we have to slightly change our

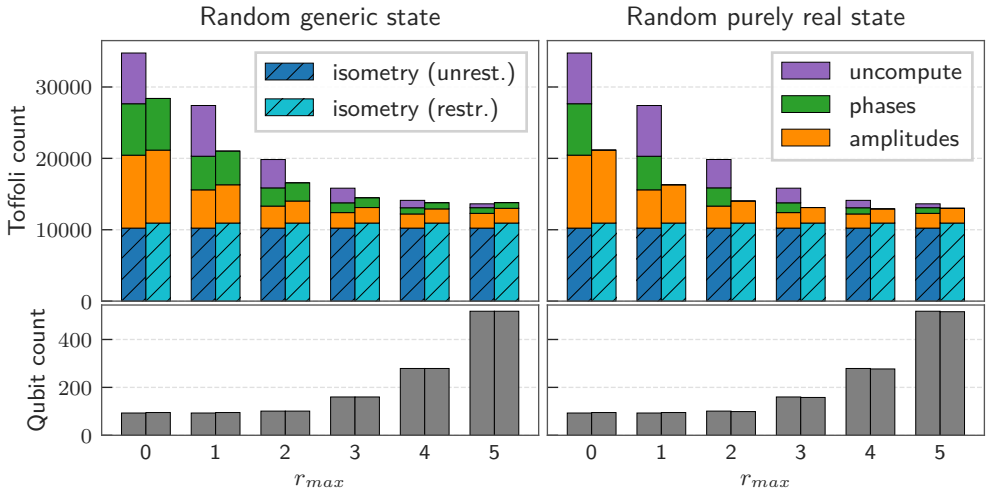


Figure 9: Toffoli and qubit counts of end-to-end sparse state preparation for a random complex (left) and purely real (right) s-sparse state on $n = 80$ qubits for $s = 10\,000$. We distinguish the individual contributions of the dense state preparation and consider different values of r_{max}, the maximum r available for the QROAMs. The case $r_{max} = 0$ corresponds to QROM. For the isometry both Algorithm 1 (restr.) and Algorithm 3 (unrest.) are applied. The resource count were calculated using Qualtran [29].

isometry circuit and the algorithm that builds it, because Algorithm 1 does not guarantee iterating over all basis states. Indeed, basis states might start in or be mapped into the subspace without being iterated over. To rectify this problem, we use Algorithm 3.

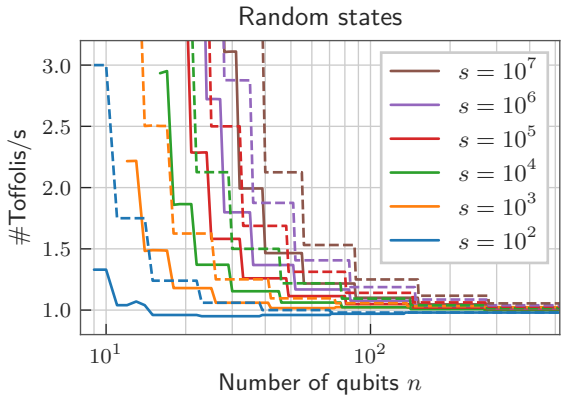
At the beginning of the algorithm, an extra ancilla qubit is set to $|1\rangle$. Moreover, the unrestricted partial unary iteration is replaced by the restricted version, and the extra qubit of a basis state is set to $|0\rangle$ when a partial unary iteration iterates over the subspace-register content of that basis state. If elements are in the subspace without having been part of a batch, we use the extra qubit to map them out of the batch and continue with the algorithm. To see that this guarantees that all basis states are iterated over, note that if there are multiple states with the same subspace-register content and one of these states is part of a partial unary iteration that sets the extra qubit from $|1\rangle$ to $|0\rangle$, then the same occurs for the others. Each of those states has some qubit outside the subspace register and the extra qubit set to $|1\rangle$, and we must ensure that this remains true until the state is iterated over in a PUI where the sign is fixed. One can easily check that this is indeed the case by noting that the only operation that can zero the non-subspace-register qubits of a state is the PUI. One can either use an extra ancilla qubit set to $|1\rangle$ for applying the Z gate, or, by keeping track of the sign values throughout the algorithm, use one of the conditionally clean qubits [32] that can be temporarily transformed to $|1\rangle$. The effect on the Toffoli count of outsourcing the uncomputation to the isometry for a random state can be seen in Figure 9 (left).

If the state to prepare, $|\Theta\rangle$, is real, the phases to encode in the last step of the dense-state-preparation circuit (dotted box in Figure 8, green boxes in Figure 9) are ± 1 signs. It is therefore possible to remove this step and, analogously to the sign fix-up, encode the phase information in the isometry. Since the unrestricted partial unary iteration is replaced by a restricted one, the bound for the Toffoli cost of the adapted isometry circuit is slightly increased to

$$\left\lceil \frac{s}{m} \right\rceil (2m + \log(\tilde{s}/m) - 3)$$

as can be seen from equations (3) and (8). Since all states are iterated over, the last term in (3) can be ignored as there is no need for the fix-up operation explained in Section 2. This estimate, together with all other estimates from the paper and some from the literature, may be found in Table 1.

In practice, as can be seen for random states on the left in Figure 10, the Toffoli cost of the restricted version of the isometry goes to s for sufficiently large n, analogous to the unrestricted



s	Time (seconds)	
	unrestr.	rest.
10^2	0.0004	0.0003
10^3	0.0043	0.0037
10^4	0.0897	0.0981
10^5	4.5485	4.7399
10^6	26.056	22.356
10^7	1488.7	813.22

Figure 10: (Left) Toffoli-cost comparison of Algorithm 1 (solid) and Algorithm 3 (dashed) on random s-sparse states with n qubits, divided by s. (Right) Wallclock-times for running implementations of Algorithm 1 (unrestr.) and Algorithm 3 (rest.) on s random basis states averaged over different values of n. The times were measured on a CPU-node with two AMD EPYC 7452. The implementations use a single thread for $s \leq 10^5$ and multi-threading for larger s. The multi-threaded version of the restricted algorithm is faster than the unrestricted version due to the way multi-threading is implemented.

version. However, for a given s there is a regime of n where the restricted isometry can require substantially more Toffoli gates than the unrestricted version. For a given state to prepare, it is therefore sensible to consider both options, taking into account the number of available qubits, to find the most suitable sparse-state-preparation circuit. In Figure 9, this is done for both a random generic (complex) and a purely real state.

5 Conclusion & Outlook

In this work, we have developed algorithms for sparse state preparation with improved Toffoli cost. We achieve this by refining an algorithm of Malvetti et al. [20] for finding and implementing isometries from the full space into a subspace whose qubit number is logarithmic in the number of states to prepare, replacing groups of individual multi-controlled-X gates with more efficient primitives we call partial unary iterations. The resulting quantum circuits are, to the best of our knowledge, the most efficient sparse state preparation circuits in terms of Toffoli and qubit counts in the literature.

We introduce two simple modifications of the well-known unary iteration circuit [23], which we call restricted and unrestricted partial unary iterations, that might be of independent interest. They iterate over intervals of values of an address register and apply unitaries on a target register controlled on the interval values. In the unrestricted version, an improved Toffoli cost is achieved by allowing the circuit to change the target register content for address register values greater than the biggest value in the interval. This is acceptable for states where one knows that no components with such address register content are present, or, as in our case, the address register values are iterated over from left to right in successive intervals, allowing prior unwanted unitaries applied to the target register to be undone in a later interval.

Finally, we review methods for implementing the dense state preparation step that constitutes the second part of the sparse state preparation algorithm and present some optimizations for the joint cost of the overall algorithm, particularly in the case of real target states. This is achieved by outsourcing the uncomputation of the angle-register and, for real target states, the encoding of the phase information to the isometry circuit, requiring some changes in the isometry.

Following the standard approach for resource estimates for fault-tolerant algorithms, the discussion in this paper is focused on the number of Toffoli gates as the figure of merit. As recent advances in generating magic states using cultivation [18, 34] and advanced distillation techniques [35] considerably lower the time and space footprint of magic-state creation, the number of Toffoli gates within an algorithm might lose its status as the main proxy for the algorithm's runtime [36]. There-

fore, in the future, adapted metrics for measuring the expected runtime of algorithms [37, 38] will be needed, and the cost of existing algorithms reevaluated.

Staying within the Toffoli-and-qubit-count approach to resource estimation, the question of whether sparse state preparation can achieve Toffoli cost sublinear in s in the practically relevant regime of limited ancilla-qubit numbers remains open.

The developed state-preparation circuits, together with the required sub-circuits, are available as Qualtran-Bloqs [30], with the computationally more expensive classical algorithms for finding the isometry outsourced to SIMD-accelerated, multi-threaded Rust code. While the runtime of these implementations allows us to find isometry circuits for sparse states with $s = 10^7$ non-zero basis states in a few minutes on reasonable hardware, we believe that there is room for improvement, potentially via efficient GPU code. Achieving such speed-ups is necessary for running the algorithms on states with $s > 10^7$ in an affordable amount of time.

In conclusion, we expect our sparse state preparation scheme to be useful as an efficient primitive in fault-tolerant quantum algorithms, such as quantum simulation and linear system solvers.

Note: After this work appeared as a preprint, the question of whether sparse state preparation with Toffoli cost sublinear in s is possible was answered affirmatively by [39, 40], which provide algorithms with Toffoli count scaling as $O(\sqrt{s})$, exploiting Toffoli/ancilla trade-offs.

Acknowledgements

This work was funded by the Ministry of Economic Affairs, Labour and Tourism Baden Württemberg through the Competence Center Quantum Computing Baden-Württemberg (KQCBW). We thank Benjamin Desef for valuable discussions and comments on the manuscript. The circuit diagrams in the paper were created with the `yquant` package [41].

Author contributions

F.R. developed the idea, implemented it, and wrote the manuscript. S.W. supervised the work and provided feedback on the manuscript. Open-weight LLMs were used for language checking and for minor assistance with coding.

References

[1] Morales, Mauro E. S. and Pira, Lirandë and Schleich, Philipp and Koor, Kelvin and Costa, Pedro C. S. and An, Dong and Aspuru-Guzik, Alán and Lin, Lin and Rebentrost, Patrick and Berry, Dominic W. "Quantum linear system solvers: A survey of algorithms and applications". Rev. Mod. Phys. **98**, 025005 (2026).

[2] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "An introduction to quantum machine learning". Contemporary Physics **56**, 172–185 (2015).

[3] Dominic W. Berry, Yu Tong, Tanuj Khattar, Alec White, Tae In Kim, Guang Hao Low, Sergio Boixo, Zhiyan Ding, Lin Lin, Seunghoon Lee, Garnet Kin-Lic Chan, Ryan Babbush, and Nicholas C. Rubin. "Rapid Initial-State Preparation for the Quantum Simulation of Strongly Correlated Molecules". PRX Quantum **6**, 020327 (2025).

[4] Stepan Fomichev, Kasra Hejazi, Modjtaba Shokrian Zini, Matthew Kiser, Joana Fraxanet, Pablo Antonio Moreno Casares, Alain Delgado, Joonsuk Huh, Arne-Christian Voigt, Jonathan E. Mueller, and Juan Miguel Arrazola. "Initial State Preparation for Quantum Chemistry on Quantum Computers". PRX Quantum **5**, 040339 (2024).

[5] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. "Trading T gates for dirty qubits in state preparation and unitary synthesis". Quantum **8**, 1375 (2024).

[6] Xiaoming Sun, Guojing Tian, Shuai Yang, Pei Yuan, and Shengyu Zhang. "Asymptotically Optimal Circuit Depth for Quantum State Preparation and General Unitary Synthesis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**, 3301–3314 (2023).

[7] Xiao-Ming Zhang, Tongyang Li, and Xiao Yuan. "Quantum State Preparation with Optimal Circuit Depth: Implementations and Applications". Phys. Rev. Lett. **129**, 230504 (2022).
[8] Kaiwen Gui, Alexander M. Dalzell, Alessandro Achille, Martin Suchara, and Frederic T. Chong. "Spacetime-Efficient Low-Depth Quantum State Preparation with Applications". Quantum **8**, 1257 (2024).
[9] C. Schön, E. Solano, F. Verstraete, J. I. Cirac, and M. M. Wolf. "Sequential Generation of Entangled Multiqubit States". Phys. Rev. Lett. **95**, 110503 (2005).
[10] Daniel Malz, Georgios Styliaris, Zhi-Yuan Wei, and J. Ignacio Cirac. "Preparation of Matrix Product States with Log-Depth Quantum Circuits". Phys. Rev. Lett. **132**, 040404 (2024).
[11] Kevin C. Smith, Abid Khan, Bryan K. Clark, S.M. Girvin, and Tzu-Chieh Wei. "Constant-Depth Preparation of Matrix Product States with Adaptive Quantum Circuits". PRX Quantum **5**, 030344 (2024).
[12] Rui Mao, Guojing Tian, and Xiaoming Sun. "Toward optimal circuit size for sparse quantum state preparation". Phys. Rev. A **110**, 032439 (2024).
[13] Niels Gleinig and Torsten Hoefler. "An Efficient Algorithm for Sparse Quantum State Preparation". In 58th ACM/IEEE Design Automation Conference (DAC). Pages 433–438. (2021).
[14] Debora Ramacciotti, Andreea I. Lefterovici, and Antonio F. Rotundo. "Simple quantum algorithm to efficiently prepare sparse states". Phys. Rev. A **110**, 032609 (2024).
[15] Fereshte Mozafari, Giovanni De Micheli, and Yuxiang Yang. "Efficient deterministic preparation of quantum states using decision diagrams". Phys. Rev. A **106**, 022617 (2022).
[16] Lvzhou Li and Jingquan Luo. "Nearly Optimal Circuit Size for Sparse Quantum State Preparation". In 52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025). Volume 334, pages 113:1–113:19. (2025).
[17] Daniel Litinski. "Magic State Distillation: Not as Costly as You Think". Quantum **3**, 205 (2019).
[18] Craig Gidney, Noah Shutty, and Cody Jones. "Magic state cultivation: growing T states as cheap as CNOT gates" (2024). arXiv:2409.17595.
[19] Michael Beverland, Earl Campbell, Mark Howard, and Vadym Kliuchnikov. "Lower bounds on the non-Clifford resources for quantum computations". Quantum Science and Technology **5**, 035009 (2020).
[20] Emanuel Malvetti, Raban Iten, and Roger Colbeck. "Quantum Circuits for Sparse Isometries". Quantum **5**, 412 (2021).
[21] Norm M. Tubman, Carlos Mejuto-Zaera, Jeffrey M. Epstein, Diptarka Hait, Daniel S. Levine, William Huggins, Zhang Jiang, Jarrod R. McClean, Ryan Babbush, Martin Head-Gordon, and K. Birgitta Whaley. "Postponing the orthogonality catastrophe: efficient state preparation for electronic structure simulations on quantum devices" (2018). arXiv:1809.05523.
[22] Tiago M. L. de Veras, Leon D. da Silva, and Adenilton J. da Silva. "Double sparse quantum state preparation". Quantum Information Processing **21**, 204 (2022).
[23] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. "Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity". Phys. Rev. X **8**, 041015 (2018).
[24] Craig Gidney. "Halving the cost of quantum addition". Quantum **2**, 74 (2018).
[25] Yuval R. Sanders, Dominic W. Berry, Pedro C.S. Costa, Louis W. Tessler, Nathan Wiebe, Craig Gidney, Hartmut Neven, and Ryan Babbush. "Compilation of Fault-Tolerant Quantum Heuristics for Combinatorial Optimization". PRX Quantum **1**, 020312 (2020).
[26] Neil J. Ross and Peter Selinger. "Optimal ancilla-free Clifford+T approximation of z-rotations". Quantum Info. Comput. **16**, 901–953 (2016).
[27] Vadym Kliuchnikov, Kristin Lauter, Romy Minko, Adam Paetznick, and Christophe Petit. "Shorter quantum circuits via single-qubit gate approximation". Quantum **7**, 1208 (2023).
[28] Renaud Vilmart, Sunheang Ty, and Chetra Mang. "Resource-Efficient Synthesis of Sparse Quantum States" (2025). arXiv:2508.05386.
[29] Matthew P. Harrigan, Tanuj Khattar, Charles Yuan, Anurudh Peduri, Noureldin Yosri, Fionn D. Malone, Ryan Babbush, and Nicholas C. Rubin. "Expressing and Analyzing Quantum Algorithms with Qualtran" (2024). arXiv:2409.04643.
[30] Felix Rupprecht and Sabine Wölk. "Code and Assets for: Sparse Quantum State Preparation with improved Toffoli cost". Zenodo (2026).

[31] Daniel Litinski and Felix von Oppen. "Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes". Quantum **2**, 62 (2018).
[32] Tanuj Khattar and Craig Gidney. "Rise of conditionally clean ancillae for efficient quantum circuit constructions". Quantum **9**, 1752 (2025).
[33] Dominic W. Berry, Craig Gidney, Mario Motta, Jarrod R. McClean, and Ryan Babbush. "Qubitization of Arbitrary Basis Quantum Chemistry Leveraging Sparsity and Low Rank Factorization". Quantum **3**, 208 (2019).
[34] Kaavya Sahay, Pei-Kai Tsai, Kathleen (Katie) Chang, Qile Su, Thomas B. Smith, Shraddha Singh, and Shruti Puri. "Fold-transversal surface code cultivation". PRX Quantum **7**, 033006 (2026).
[35] Diego Ruiz, Jérémie Guillaud, Christophe Vuillot, and Mazyar Mirrahimi. "Unfolded distillation: very low-cost magic state preparation for biased-noise qubits". npj Quantum Information **12**, 53 (2026).
[36] William J. Huggins, Tanuj Khattar, and Nathan Wiebe. "Productionizing Quantum Mass Production" (2025). arXiv:2506.00132.
[37] Sam McArdle, Alexander M. Dalzell, Aleksander Kubica, and Fernando G. S. L. Brandão. "The Fast for the Curious: How to accelerate fault-tolerant quantum applications" (2025). arXiv:2510.26078.
[38] William J. Huggins, Tanuj Khattar, Amanda Xu, Matthew Harrigan, Christopher Kang, Guang Hao Low, Austin Fowler, Nicholas C. Rubin, and Ryan Babbush. "The FLuid Allocation of Surface code Qubits (FLASQ) cost model for early fault-tolerant quantum algorithms" (2025). arXiv:2511.08508.
[39] Tongyang Li, Fengning Ou, Xinzhao Wang, Penghui Yao, Pei Yuan, and Shengyu Zhang. "Optimal T Counts under Sparsity: from QROM to State Preparation and Block Encoding" (2026). arXiv:2607.28260.
[40] Jingquan Luo and Lvzhou Li. "Sparse Quantum State Preparation with Sublinear T-Count" (2026). arXiv:2608.00414.
[41] Benjamin Desef. "Yquant: Typesetting quantum circuits in a human-readable language" (2021). arXiv:2007.12931.

A AND gates

In Figure 11 we recall the definition of the AND gate primitive and its measurement-based adjoint as introduced in [23, 24].

Figure 11: AND gate and measurement-based inverse.

B Proof of bound

In this section, we give the inductive proof of the bound (6). For this, we let $h(x)$ denote the Hamming weight of the binary representation of a natural number x. The bound follows from the following Lemma with $k \coloneqq w - l$ and $r \coloneqq \lceil \frac{s}{m} \rceil$, for which we have

$$2^{k-1} = \frac{2^{w-1}}{2^l} = \frac{2^{w-1}}{m} \leq \left\lceil \frac{s}{m} \right\rceil \leq \frac{2^w}{m} = 2^k.$$

Lemma. *Let $k \in \mathbb{N}$. If r is a natural number with $2^{k-1} \leq r \leq 2^k$ and $C_r \coloneqq \{x \in \mathbb{N}_{<r} \mid h(x) > 1\}$, then $S_r \coloneqq \sum_{x \in C_r} (h(x) - 1) \leq r\left(\frac{k}{2} - 1\right) + 1$.*

Proof. We prove the statement by induction on k. For $k = 1$, the statement is easy to verify. For the induction step, define $\tilde{C}_r \coloneqq \{x \in C_r \mid x \geq 2^{k-1}\}$ and notice that $S_r = S_{2^{k-1}} + \sum_{x \in \tilde{C}_r} (h(x) - 1)$. Let $x \in \tilde{C}_r$ and define $y_x \coloneqq x - 2^{k-1}$. Then $h(x) = h(y_x) + 1$, and using the induction hypothesis we have

$$\begin{aligned}\sum_{x \in \tilde{C}_r} (h(x) - 1) = (r - 2^{k-1} - 1) + \sum_{x \in \tilde{C}_r} (h(y_x) - 1) &= (r - 2^{k-1} - 1) + \sum_{y \in C_{r-2^{k-1}}} (h(y) - 1) \\ &\leq (r - 2^{k-1} - 1) + (r - 2^{k-1}) \left(\frac{k-1}{2} - 1\right) + 1.\end{aligned}$$

Here we use the fact that $|\tilde{C}_r| = r - 2^{k-1} - 1$, that the elements y_x which do not belong to $C_{r-2^{k-1}}$ are precisely those with $h(y_x) = 1$, and the induction hypothesis. Putting everything together, using $2^k \geq r$ and, once more, the induction hypothesis yields

$$\begin{aligned}S_r &= S_{2^{k-1}} + \sum_{x \in \tilde{C}_r} (h(x) - 1) \\ &\leq 2^{k-1} \left(\frac{k-1}{2} - 1\right) + 1 + (r - 2^{k-1} - 1) + (r - 2^{k-1}) \left(\frac{k-1}{2} - 1\right) + 1 \\ &= r\left(\frac{k}{2} - \frac{1}{2}\right) - \frac{2^k}{2} + 1 \leq r\left(\frac{k}{2} - 1\right) + 1.\end{aligned}$$

□

C Algorithms

Algorithm 1: Finding the isometry and bijection.

Data: Tableau of n-bit bitstrings $[|C_i\rangle]_{i\in[s]}$ written as $|C_i\rangle \coloneqq |C_i'\rangle^s |C_i''\rangle^r$ with $|C_i'\rangle^s$ having $l \coloneqq \lceil \log(s) \rceil$ bits.

Result: Quantum circuit G and bijection $f\colon [s] \to S$ where $S \subseteq [2^{\lceil \log(s) \rceil}]$ such that $G|C_i\rangle = |f(i)\rangle^s |0\rangle^r$ for all $i \in [s]$.

$G \leftarrow [\,]$.
$k \leftarrow 0$.
$batch \leftarrow [\,]$. `// Row indices of batch elements in tableau.`
$m \leftarrow \max_{p\in\mathbb{N}}(2^p \mid 2^p \le n-l)$. `// Maximum batch size.`
while *True* **do**
  $b \leftarrow |batch|$. `// Current batch size.`
  if $b = m$ **then**
    $\mathrm{PUI}_{k-b}^{k-1} \leftarrow$ unrestricted PUI, mapping $|C_j\rangle = |k_j\rangle^s |e_{q_j}\rangle^r \mapsto |k_j\rangle^s |0\rangle^r$ for all $j \in batch$.
    G.append(PUI_{k-l}^{k-1}).
    $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{PUI}_{k-b}^{k-1} |C_i\rangle]_{i\in[s]}$. `// Update tableau.`
    $batch \leftarrow [\,]$.
  if $|C_i\rangle = |k_i\rangle^s |0\rangle^r$ *for all* $i \in [s] \setminus batch$, **then**
    $\widetilde{\mathrm{PUI}}_{k-b}^{k-1} \leftarrow$ restricted PUI, mapping $|C_j\rangle = |k_j\rangle^s |e_{q_j}\rangle^r \mapsto |k_j\rangle^s |0\rangle^r$ for all $j \in batch$.
    G.append($\widetilde{\mathrm{PUI}}_{k-l}^{k-1}$). `// Restricted PUI: Higher subspace states untouched.`
    $[|C_i\rangle]_{i\in[s]} \leftarrow [\widetilde{\mathrm{PUI}}_{k-b}^{k-1} |C_i\rangle]_{i\in[s]}$.
    if $|C_j\rangle = |k-1\rangle^s |e_{b-1}\rangle^r$ *for some* $j \in [s]$, **then**
      `// Both states` $|k-1\rangle^s |0\rangle^r$ `and` $|k-1\rangle^s |e_{b-1}\rangle^r$ `present before PUI.`
      $|k'\rangle^s \leftarrow$ any bitstring such that $|C_i\rangle \neq |k'\rangle^s |0\rangle^r$ for all $i \in [s]$.
      $\mathrm{CMX}_{l+b-1} \leftarrow$ multi-target CX with control $l+b-1$ mapping $|C_j\rangle \mapsto |k'\rangle^s |e_{b-1}\rangle^r$.
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{CMX}_{l+b-1} |C_i\rangle]_{i\in[s]}$.
      $\mathrm{MCX}^{l+b-1} \leftarrow$ multi-control CX with target $l+b-1$, mapping $|C_j\rangle \mapsto |k'\rangle^s |0\rangle^r$.
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{MCX}^{l+b-1} |C_i\rangle]_{i\in[s]}$.
      G.extend($[\mathrm{CMX}_{l+b-1}, \mathrm{MCX}^{l+b-1}]$).
    $f(i) \leftarrow k_i$ where $|C_i\rangle = |k_i\rangle^s |0\rangle^r$ for all $i \in [s]$.
    return G, f.
  if *there exists* $j \in [s] \setminus batch$ *such that* $|C_j''\rangle^r \neq |0\rangle^r$, **then**
    if $r \neq l+b$, **then**
      G.append(SWAP_r^{l+b}). `// Swap qubits` r `and` $l+b$.
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{SWAP}_r^{l+b} |C_i\rangle]_{i\in[s]}$.
  else
    $j \leftarrow$ any $j \in [s] \setminus batch$ such that $|C_j''\rangle^r \neq |0\rangle^r$.
    $u \leftarrow$ any bit-index with $l \le u < l+b$ at which $|C_j\rangle$ is non-zero.
    $v \leftarrow$ any bit-index at which $|C_j\rangle$ and $|C_w\rangle = |k_w\rangle^s |e_{u-l}\rangle^r$ from the batch differ.
      `//` $|C_w\rangle$ `in batch by construction.`
    if $|C_j\rangle$ *is zero at* v **then**
      G.append($\mathrm{Toff}_{u,\bar{v}}^{l+b}$). `// Toffoli; negative control` v.
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{Toff}_{u,\bar{v}}^{l+b} |C_i\rangle]_{i\in[s]}$.
    else
      G.append($\mathrm{Toff}_{u,v}^{l+b}$).
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{Toff}_{u,v}^{l+b} |C_i\rangle]_{i\in[s]}$.
  $\mathrm{CMX}_{l+b} \leftarrow$ multi-target CX with control $l+b$ mapping $|C_j\rangle \mapsto |k\rangle^s |e_b\rangle^r$.
  G.append(CMX_{l+b}).
  $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{CMX}_{l+b} |C_i\rangle]_{i\in[s]}$.
  $batch$.append(j).
  $k \leftarrow k+1$.

Algorithm 2: Finding the circuit and mapping for unrestricted partial unary iteration.

Data: Interval $S \coloneqq [|l\rangle^a, |r\rangle^a]$ of address basis states from a w-qubit address register and corresponding quantum gates $U_l, \ldots, U_r$ to be applied on a target register.

Result: Quantum circuit G, subset $L \subseteq \{r+1, \ldots, 2^w - 1\}$ and mapping $g\colon L \to \{l, \ldots, r\}$ such that $G = \sum_{i=l}^{r} |i\rangle\langle i|^a \otimes U_i + \sum_{i \in L} |i\rangle\langle i|^a \otimes U_{g(i)}$.

$G \leftarrow [\,]$.
$I \leftarrow [|0\rangle^a, |2^w - 1\rangle^a]$.
`UPUI(`I, *None*, $\emptyset$`)`.

def `UPUI(`*interval, control, free*`)`**:**
- $d \leftarrow w - \log(|interval|)$. `// Current address qubit; big-endian.`
- **if** *interval* $= [|i\rangle^a, |i\rangle^a]$, **then**
  - **if** *control is not None,* **then**
    - G.append($\mathrm{C}_{control}U_i$). `//` U_i `controlled on` *control*.
  - **else**
    - G.append(U_i).
  - **if** $|free| > 0$, **then**
    - $J \leftarrow$ integers with w-bit binary representation equal to i on indices not in $free$ and at least one 1 at some position in $free$.
    - $f(j) \leftarrow i$ for all $j \in J$.
  - **return**
- $I_l, I_r \leftarrow$ left and right half-interval of *interval*.
- **if** $I_l \cap S \neq \emptyset$ *and* $I_r \cap S \neq \emptyset$, **then**
  - **if** *control is not None,* **then**
    - G.append($\mathrm{AND}^{anc}_{control,\bar{d}}$). `// Negative control` d; *anc* `is new ancilla.`
    - `UPUI(`$I_l, anc, free$`)`.
    - G.append($\mathrm{CX}^{anc}_{control}$).
    - `UPUI(`$I_r, anc, free$`)`.
    - G.append($\mathrm{AND}^{anc}_{control,\bar{d}}{}^{\dagger}$).
  - **else**
    - G.append(X_d). `//` X `gate on` d.
    - `UPUI(`$I_l, d, free$`)`.
    - G.append(X_d).
    - `UPUI(`$I_r, d, free$`)`.
- **else if** $I_l \cap S \neq \emptyset$ *and* $I_r \cap S = \emptyset$, **then**
  - `UPUI(`$I_l, control, free \cup \{d\}$`)`.
- **else** `// Must have` $I_l \cap S = \emptyset$ `and` $I_r \cap S \neq \emptyset$.
  - **if** *control is not None,* **then**
    - G.append($\mathrm{AND}^{anc}_{control,d}$). `//` *anc* `is new ancilla.`
    - `UPUI(`$I_r, anc, free$`)`.
    - G.append($\mathrm{AND}^{anc}_{control,d}{}^{\dagger}$).
  - **else**
    - `UPUI(`$I_r, d, free$`)`.

Algorithm 3: Finding the isometry and bijection with phase fixing.

Data: Tableau of n-bit bitstrings $[|C_i\rangle]_{i\in[s]}$ written as $|C_i\rangle \coloneqq |C_i'\rangle^s |C_i''\rangle^r$ with $|C_i'\rangle^s$ having $l \coloneqq \lceil \log(s) \rceil$ bits. List of ± 1-signs $[a_i]_{i\in[s]}$.

Result: Quantum circuit G and bijection $f\colon [s] \to [s]$ such that $G|C_i\rangle = a_i |f(i)\rangle^s |0\rangle^r$ for all $i \in [s]$.

$G \leftarrow [\,]$.
$k \leftarrow 0$.
$batch \leftarrow [\,]$.
$m \leftarrow \max_{p\in\mathbb{N}}(2^p \mid 2^p \le n - l)$
$[|D_i\rangle^e]_{i\in[s]} \leftarrow [|1\rangle^e]_{i\in[s]}$. `// Extend tableau with a` $|1\rangle$`-column as` $(n+1)$`-th qubit.`
while *True* **do**
  $b \leftarrow |batch|$.
  if $b = m$ *or* $k = s$, **then**
    $\widetilde{\mathrm{PUI}}_{k-b}^{k-1} \leftarrow$ restricted PUI, mapping $|C_j\rangle |D_j\rangle^e = |k_j\rangle^s |e_{q_j}\rangle^r |D_j\rangle^e \mapsto a_j |k_j\rangle^s |0\rangle^r |0\rangle^e$ for all $j \in batch$. `// Fix sign; set extra qubit to` $|0\rangle$`.`
    G.append($\widetilde{\mathrm{PUI}}_{k-b}^{k-1}$).
    $[(a_i, |C_i\rangle, |D_i\rangle^e)]_{i\in[s]} \leftarrow [\widetilde{\mathrm{PUI}}_{k-b}^{k-1}(a_i, |C_i\rangle, |D_i\rangle^e)]_{i\in[s]}$. `// Update all tableaus.`
    $batch \leftarrow [\,]$.
    if $k = s$, **then**
      $f(i) \leftarrow k_i$ where $|C_i\rangle = |k_i\rangle^s |0\rangle^r$ for all $i \in [s]$.
      return G, f.
  if *there exists* $j \in [s]$ *such that* $|C_j\rangle$ *is non-zero at bit* $r \ge l + b$, **then**
    if $r \ne l + b$, **then**
      G.append(SWAP_r^{l+b}).
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{SWAP}_r^{l+b} |C_i\rangle]_{i\in[s]}$.
  else if *there exists* $j \in [s] \setminus batch$ *such that* $|C_j''\rangle^r \ne |0\rangle^r$, **then**
    $u \leftarrow$ any bit-index with $l \le u < l + b$ at which $|C_j\rangle$ is non-zero.
    $v \leftarrow$ any bit-index at which $|C_j\rangle$ and $|C_w\rangle = |k_w\rangle^s |e_{u-l}\rangle^r$ in the batch differ.
    if $|C_j\rangle$ *is zero at* v **then**
      G.append($\mathrm{Toff}_{u,\bar{v}}^{l+b}$).
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{Toff}_{u,\bar{v}}^{l+b} |C_i\rangle]_{i\in[s]}$.
    else
      G.append($\mathrm{Toff}_{u,v}^{l+b}$).
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{Toff}_{u,v}^{l+b} |C_i\rangle]_{i\in[s]}$.
  else
    $j \leftarrow$ any $j \in [s] \setminus batch$ such that $|D_j\rangle^e = |1\rangle^e$. `// Exists as` $k < s$.
    for $l \in batch \setminus \{j\}$ *with* $|D_l\rangle^e = |1\rangle^e$ **do**
      G.append($\mathrm{CX}_{q_l}^{l+b}$) where $|C_l\rangle = |k_l\rangle^s |e_{q_l}\rangle^r$. `// Counter effect of later CX.`
      $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{CX}_{q_l}^{l+b} |C_i\rangle]_{i\in[s]}$.
    G.append(CX_n^{l+b}) `// CX gate with control` n `and target` $l + b$.
    $[(|C_i\rangle, |D_i\rangle^e)]_{i\in[s]} \leftarrow [\mathrm{CX}_n^{l+b}(|C_i\rangle, |D_i\rangle^e)]_{i\in[s]}$.
  $\mathrm{CMX}_{l+b} \leftarrow$ multi-target CX with control $l + b$ mapping $|C_j\rangle \mapsto |k\rangle^s |e_b\rangle^r$.
  G.append(CMX_{l+b}).
  $[|C_i\rangle]_{i\in[s]} \leftarrow [\mathrm{CMX}_{l+b} |C_i\rangle]_{i\in[s]}$.
  $batch$.append(j).
  $k \leftarrow k + 1$.

D Resource costs for other sparse quantum state preparation algorithms

Most prior papers in the literature proposing algorithms for sparse quantum state preparation did so using the asymptotic circuit size as their figure of merit. As we are interested in constant factor Toffoli gate counts and ancilla numbers we briefly look at the corresponding resource estimates for the respective papers. References to steps, algorithms or equations always refer to the corresponding paper under consideration.

We start with the group of papers [12, 13, 21, 22] in which the sparse state is built iteratively one basis state with its corresponding coefficient at a time. As mentioned in the introduction, each of the s basis states requires the implementation of a multi-controlled rotation gate or an equivalent construction which may always be reduced to a controlled rotation and a multi-controlled X gate [24]. The cost of the controlled rotation, as explained in Section 1, is given by $\lceil \log(s/\epsilon) \rceil$. Here we look at the number of control qubits for the multi-controlled X gates. Using the standard AND construction [23, 24] a w-controlled X may be implemented with $w-1$ Toffolis and $w-2$ ancillas.

- [12]: The BE-QRAM algorithms works in batches of computational basis states of size $\lceil s/k \rceil$ with $k := \lfloor \log(n) - \log(\log(n)) \rfloor$. The number of control qubits for each state (Step 3) is $2^k + 1$. Moreover, each batch has two $(n - 2^k)$-controlled X gates (Steps 2 and 4).

- [13]: The control qubits are found by a halving procedure leading to at most $\lceil \log(d) \rceil$ controls.

- [21]: Declared cost of $O(n)$ without details. Assume an n-controlled X gate.

- [22]: The controls in the CVO-QRAM algorithm are the qubits on which the current basis state is $|1\rangle$. Summed over all s states the worst case control qubit number, i.e., the sum of the Hamming weights, is lower bounded by $ns/2$.

We now turn to the second group of algorithms [4, 14, 16, 20] in which first a dense state is prepared which is then transformed into the sparse target state via an isometry. Here we look at the Toffoli and qubit costs of the isometry. In the paper [4] the figures of merit are already Toffoli and qubit gate counts. Moreover, the approach in [20] was investigated in detail in Section 2.

- [14]: The isometry is a permutation of basis states which may be decomposed into cycles (Section IV). The sum of the lengths of the cycles is upper bounded by $2s$, which is the case if there are s transpositions, i.e., cycles of length two. The non-Clifford cost for the implementation of a cycle of length w is given by $w+1$ multi-controlled X gates with n controls (Appendix B). The approach uses an extra ancilla qubit. In total the worst case Toffoli cost of the algorithm is $3s(n-1)$ with ancilla cost $n-1$.

- [16]: The permutation to implement can be chosen to consist of at most s disjoint transpositions (Algorithm 1). It is implemented in m batches with m being a power of two. Each transposition requires two $\lceil \log(2m) \rceil$-controlled X gates (Steps 2a and 2c in the proof of Lemma 12). Moreover each batch uses a multi-controlled X gate with $n - \lfloor \log(2m) \rfloor$ controls (Step 2b). The total sum of Toffoli gates is therefore

$$\lceil s/m \rceil (n - \lfloor \log(2m) \rfloor - 1 + 2m(\lceil \log(2m) \rceil - 1)).$$

  Contrary to the paper where m is set to $2^{\lfloor \log(\log(m)/4) \rfloor}$ for optimizing the circuit size, the Toffoli cost is optimal for $m := s$, where there is only a single batch. The cost is then

$$2s(\lceil \log(2s) \rceil - 1) + n - \lfloor \log(2s) \rfloor - 1$$

  with ancilla count $\max(n - \lfloor \log(2s) \rfloor, \lceil \log(2s) \rceil) - 1$. As we usually assume that $s \ll 2^n$, we use $n - \lfloor \log(2s) \rfloor$ in Table 1. For brevity, we also drop the -1 in the Toffoli count there.