

A PRACTICAL QUANTUM ALGORITHM FOR THE SCHUR TRANSFORM

WILLIAM M. KIRBY^a

*Physics Department, Williams College
Williamstown, MA 01267, USA*

FREDERICK W. STRAUCH^b

*Physics Department, Williams College
Williamstown, MA 01267, USA*

Received September 25, 2017

Revised July 27, 2018

We describe an efficient quantum algorithm for the quantum Schur transform. The Schur transform is an operation on a quantum computer that maps the standard computational basis to a basis composed of irreducible representations of the unitary and symmetric groups. We simplify and extend the algorithm of Bacon, Chuang, and Harrow, and provide a new practical construction as well as sharp theoretical and practical analyses. Our algorithm decomposes the Schur transform on n qubits into $O(n^4 \log(\frac{n}{\epsilon}))$ operators in the Clifford+T fault-tolerant gate set and uses exactly $2\lfloor \log_2(n) \rfloor - 1$ ancillary qubits. We extend our qubit algorithm to decompose the Schur transform on n qudits of dimension d into $O(d^{1+p} n^{3d} \log^p(\frac{dn}{\epsilon}))$ primitive operators from any universal gate set, for $p \approx 3.97$.

Keywords:

Communicated by: R Cleve & J Eisert

1 Introduction

The *Schur transform* is a useful routine in quantum computing. It is a change of basis on a register of qudits, from the computational basis (composed of tensor products of the individual qudits' states) to an alternate basis called the *Schur basis* [1]. For qubits (dimension $d = 2$), the Schur basis is composed of eigenvectors of the global spin of the whole register. For qudits, the situation is more complex, but in both cases the Schur basis exhibits global rather than local symmetry. The Schur transform generalizes the more familiar *Clebsch-Gordan* (CG) transform, which performs this operation on two subsystems.

We can describe the Schur basis more rigorously. If \mathcal{C}^d is the Hilbert space for a qudit of dimension d , we can write the Hilbert space for a register of n such qudits as $(\mathcal{C}^d)^{\otimes n} = \underbrace{\mathcal{C}^d \otimes \mathcal{C}^d \otimes \dots \otimes \mathcal{C}^d}_{n \text{ copies}}$. The Schur basis is a decomposition of $(\mathcal{C}^d)^{\otimes n}$ into irreducible modules

(irreps) of the unitary group \mathcal{U}_d and the symmetric group \mathcal{S}_n .

^awmk1@williams.edu

^bfws1@williams.edu

In this paper, we describe an efficient and practical quantum algorithm for the Schur transform. For n qubits, our algorithm decomposes the Schur transform into $O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right)$ Clifford+T operators, which can be implemented fault-tolerantly [2, 3]. We refer to this number as the *sequence length*: it is the “quantum runtime” of our algorithm, since each of the operators in the sequence will need to be performed in order. We also extend our algorithm to n qudits of dimension d , for which it decomposes the Schur transform into a sequence of $O\left(d^{1+p} n^{3d} \log^p\left(\frac{dn}{\epsilon}\right)\right)$ primitive operators from any universal gate set (for $p \approx 3.97$). Our qubit algorithm employs exactly $2\lceil\log_2(n)\rceil - 1$ ancillary qubits. This work simplifies and extends the work by Bacon, Chuang, and Harrow (BCH) in [1, 4]: in particular, we provide a practical implementation and explicit analysis of a modification of the qubit algorithm [4] using a minimum of ancilla, and our framework can be extended to a qudit algorithm that expands upon arguments in Section V of [1].

Throughout this paper, we primarily focus on the special case of qubits ($d = 2$), both because it is helpful in developing the right intuitions and pictures of the Schur transform, especially for readers with a background in physics, and because ultimately we provide an explicit implementation of our algorithm for the qubit case. The paper is organized as follows: In the remainder of this section, we describe the specific mathematical background for the Schur transform. We assume some general knowledge of the representation theory of Lie groups and the symmetric group; [5] is a good reference for this material. In §2, we describe the Clebsch-Gordan transform, which we will use recursively to construct the Schur transform. In §3, we describe our qubit algorithm for the Schur transform, and in §4, we analyze its runtime. In §5, we describe how to extend our qubit algorithm to an algorithm for qudits of arbitrary dimension, and provide an analysis for this general case. In §6, we summarize and compare our work with that of BCH.

1.1 *Mathematical background*

A group G is *reductive* if every (regular) representation of G is either irreducible or completely reducible.

Theorem 1 (Isotypic decomposition [1]) *Let G be a reductive group. Then for any G -module V ,*

$$V \cong \bigoplus_{i=1}^k V^{(i)} \tag{1}$$

for some irreps $V^{(i)}$.

Proof. See [5] \square .

Note that the $V^{(i)}$ need not all be inequivalent; we will discuss multiplicities shortly. One can show that the unitary group \mathcal{U}_d , as well as any finite group, is reductive (see [5]). These facts prove the following theorem:

Theorem 2 *Any module of the symmetric group \mathcal{S}_n (since it is finite), or the unitary group \mathcal{U}_d , can be decomposed into a finite direct sum of irreducible submodules, as in (1).*

From theorem 2 we obtain the Schur transform. We denote the general linear group acting on $(\mathcal{C}^d)^{\otimes n}$ (the Hilbert space of a register of n d -dimensional qudits) by $GL((\mathcal{C}^d)^{\otimes n})$. To describe the Schur transform, consider the following pair of representations:

- $P : S_n \rightarrow GL((\mathcal{C}^d)^{\otimes n})$, defined by

$$P(s)|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle = |\phi_{s^{-1}(1)}\rangle \otimes |\phi_{s^{-1}(2)}\rangle \otimes \cdots \otimes |\phi_{s^{-1}(n)}\rangle \tag{2}$$

where s is any permutation (in S_n); that is, $P(s)$ permutes the component states according to s .

- $Q : \mathcal{U}_d \rightarrow GL((\mathcal{C}^d)^{\otimes n})$, defined by

$$Q(U)|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle = U|\phi_1\rangle \otimes U|\phi_2\rangle \otimes \cdots \otimes U|\phi_n\rangle \tag{3}$$

for any $U \in \mathcal{U}_d$; that is, $Q(U)$ applies U to each qudit.

By theorem 2, $(\mathcal{C}^d)^{\otimes n}$ can be decomposed into a finite direct sum of irreducible submodules of either of these representations. Let the irreducible Q -modules be denoted \mathcal{Q}_λ^d , for some index λ , and let the irreducible P -modules be denoted \mathcal{P}_λ . Then we can write

$$(\mathcal{C}^d)^{\otimes n} \stackrel{\mathcal{U}_d}{\cong} \bigoplus_{\lambda} \mathcal{M}_\lambda \otimes \mathcal{Q}_\lambda^d \tag{4}$$

and

$$(\mathcal{C}^d)^{\otimes n} \stackrel{S_n}{\cong} \bigoplus_{\lambda} \mathcal{N}_\lambda \otimes \mathcal{P}_\lambda \tag{5}$$

where \mathcal{M}_λ and \mathcal{N}_λ are multiplicity spaces. These expressions can be combined by applying *Schur duality* [1], which states that the multiplicity spaces \mathcal{M}_λ are isomorphic to the irreps \mathcal{P}_λ (and equivalently, the multiplicity spaces \mathcal{N}_λ are isomorphic to the irreps \mathcal{Q}_λ^d). Therefore, we can consolidate our two expressions (4) and (5) into one:

$$(\mathcal{C}^d)^{\otimes n} \stackrel{\mathcal{U}_d \times S_n}{\cong} \bigoplus_{\lambda} \mathcal{P}_\lambda \otimes \mathcal{Q}_\lambda^d \tag{6}$$

A derivation of this result can be found in [1].

We have not yet stated what the index λ is, nor how to find the irreps \mathcal{P}_λ and \mathcal{Q}_λ^d . We now give these results, referring the reader to other sources for their derivations.

The index λ runs over all *partitions* of n [1]. For λ to partition n means that $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_\ell)$ for positive integers λ_i such that $\sum_{i=1}^{\ell} \lambda_i = n$ and $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_\ell > 0$. We call ℓ the *degree* of λ , and we write $\lambda \vdash n$. There is one distinct irrep of \mathcal{U}_d and one distinct irrep of S_n for each $\lambda \vdash n$.

To obtain the dimensions of the irreps, we use Young diagrams and tableaux. The *Young diagram* associated to a partition λ is an array of boxes, with λ_i boxes in the i th row. For example, if $\lambda = (3, 1)$, then the Young diagram of λ is

$$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & & \\ \hline \end{array} \tag{7}$$

A *tableau* (plural: *tableaux*) is an assignment of integers to the boxes in a Young diagram. We call a tableau associated to some partition λ a λ -tableau. Here are a few (3, 1)-tableaux

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & & \\ \hline \end{array}, \quad \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 2 & & \\ \hline \end{array}, \quad \begin{array}{|c|c|c|} \hline 4 & 4 & 3 \\ \hline 1 & & \\ \hline \end{array} \tag{8}$$

A *standard tableau* is a tableau whose entries are strictly increasing across the rows and down the columns: for example, the first tableau in (8) is a standard $(3, 1)$ -tableau; we further restrict the entries to be the set $\{1, 2, \dots, n\}$. This last restriction is not part of the usual definition of a standard tableau, but it will simplify our usage. A *semistandard tableau* is a tableau whose entries are weakly increasing across the rows and strictly increasing down the columns: for example, the first and second tableaux in (8) are semistandard $(3, 1)$ -tableaux.

We find the dimensions of our irreps by counting λ -tableaux:

$$\dim(\mathcal{Q}_\lambda^d) = \# \text{ of semistandard } \lambda\text{-tableaux with entries in } \{1, 2, \dots, d\} \quad (9)$$

$$\dim(\mathcal{P}_\lambda) = \# \text{ of standard } \lambda\text{-tableaux} \quad (10)$$

(9) has the following useful corollary:

Corollary 1 *For $\lambda \vdash n$ (for some integer n), if $\deg(\lambda) > d$, then $\dim(\mathcal{Q}_\lambda^d) = 0$.*

Our critical result in this section was (6). We define the *Schur basis* to be a basis in which (6) is an equality. The *Schur transform* is a change of basis from the computational basis to the Schur basis. Since we know from (6) that the \mathcal{S}_n -irreps \mathcal{P}_λ act as the multiplicity spaces for the \mathcal{U}_d -irreps \mathcal{Q}_λ^d and vice versa, we know that whether we structure our implementation in terms of (4) or (5), we will always obtain (6). Through most of this document, we will work in terms of (4). Thus, since the irreps \mathcal{P}_λ are the multiplicity spaces for the irreps \mathcal{Q}_λ^d in (6), by (10) we have

$$\text{multiplicity of } \mathcal{Q}_\lambda^d = \# \text{ of standard } \lambda\text{-tableaux} \quad (11)$$

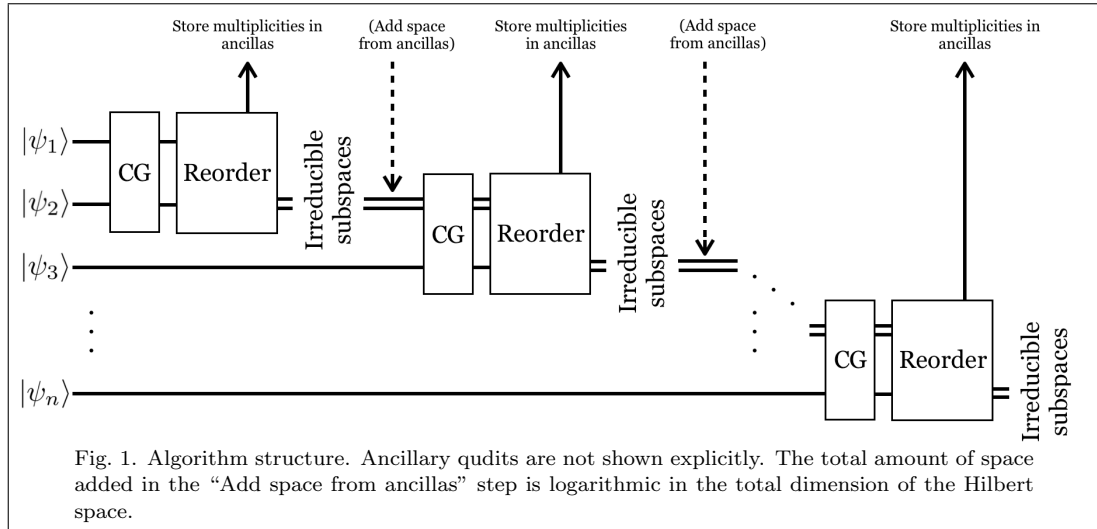
1.2 Applications and previous work

A few of the better known applications of the Schur transform are decoherence-free encoding [6, 7, 8], quantum hypothesis testing [9, 10], spectrum estimation [11, 12], entanglement concentration [13, 14], and reference-frame independent quantum communication [15].

David Bacon, Isaac Chuang, and Aram Harrow published a pair of papers [1, 4] in which they first described efficient quantum algorithms for the Schur transform. Their work provides an analysis of the circuit and its runtime, showing that the runtime is polynomial in n , but without finding explicit exponents. A subsequent paper [16] describes a generalization of BCH's construction, but similarly does not provide a constructive algorithm or an explicit sequence length.

Our algorithm is inspired by the work of BCH and shares its outermost layer of structure with their construction: it recursively decomposes the Schur transform into a succession of operators built out of Clebsch-Gordan (CG) transforms (see Fig. 1). Our algorithm differs from that of BCH in the implementation of each individual recursion step. The insight that motivates our implementation of the recursion step goes roughly as follows:

At each large-scale step in the algorithm, we have a decomposition of the full Hilbert space into some set of subspaces, each of which has an irrep in the decomposition (6) for some subset of the qudits in the register. We will show herein that a careful reordering of these subspaces at each step simplifies the algorithm in such a way that an explicit analysis is possible. The reordering also makes the structure of the algorithm directly reflect the structure of the underlying mathematical objects. To perform the reordering, we add a logarithmic number of ancillary qudits to the register. Then, roughly speaking, the reordering arranges the



irreducible representations of the globally-symmetric unitary group (for the current iteration) over the states of the ancillary qudits, which then act as indices. This allows the next recursion step to act in parallel on each of the distinct irreducible representations, whose individual dimensions are logarithmic in the total dimension, which in turn allows the algorithm to achieve polynomial instead of exponential runtime. This way of structuring of the algorithm is one of the main contributions of our work, as it allows for a relatively simple way to encode the irrep labels with a minimal number of ancilla. We will discuss this approach in detail in §3 (for qubits) and §5 (for qudits).

2 The Clebsch-Gordan Transform

In this section we describe the *Clebsch-Gordan* (CG) transform, which will form the recursion step in our construction of the Schur transform in the next section.

2.1 Mathematical perspective

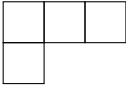
For $\mu \vdash n$ and $\nu \vdash m$, let \mathcal{Q}_μ^d and \mathcal{Q}_ν^d be irreps of \mathcal{U}_d acting on n and m qudits. By theorem 2, we can decompose the tensor product module into irreps:

$$\mathcal{Q}_\mu^d \otimes \mathcal{Q}_\nu^d \cong^{\mathcal{U}_d} \bigoplus_{\lambda \vdash (n+m)} \mathcal{M}_\lambda \otimes \mathcal{Q}_\lambda^d \tag{12}$$

where \mathcal{M}_λ is the multiplicity space associated to the irrep \mathcal{Q}_λ^d . The CG transform maps the basis on the left-hand side of (12) to the basis on the right-hand side. In the case $\nu = (1)$ (which will turn out to be the case we are interested in), (12) simplifies to the following expression:

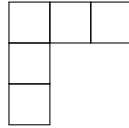
$$\mathcal{Q}_\mu^d \otimes \mathcal{Q}_{(1)}^d \cong^{\mathcal{U}_d} \bigoplus_{\lambda} \mathcal{Q}_\lambda^d \tag{13}$$

where $\lambda = \mu + \mathbf{e}_j$ for some j such that λ is a valid partition and has degree less than or equal to d [1]. We can visualize this schematically in terms of Young diagrams: for example, if

$d = 2$ and $\mu = (3, 1)$, then the corresponding Young diagram is , and (13) can be visualized as

$$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & & \\ \hline \end{array} \otimes \begin{array}{|c|} \hline \square \\ \hline \end{array} \stackrel{U_d}{\cong} \begin{array}{|c|c|c|c|} \hline \square & \square & \square & \square \\ \hline \square & & & \\ \hline \end{array} \oplus \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \tag{14}$$

If d were greater than 2, the Young diagram



would also appear on the right-hand side. Thus we can think of the version of the CG transform that we are interested in as the “add-a-box” operation, which maps the standard tensor product basis for $\mathcal{Q}_\mu^d \otimes \mathcal{Q}_{(1)}^d$ to a basis in which (13) is an equality [1].

2.2 Physical perspective

We can also present the CG transform for $d = 2$ in a context more familiar to physicists, most easily described in terms of spin.

Definition 1 The total spin operator (squared) \mathbf{J}^2 is defined by

$$\mathbf{J}^2 |j, m\rangle = j(j + 1)\hbar^2 |j, m\rangle \tag{15}$$

Definition 2 The spin projection operator J^z is defined by

$$J^z |j, m\rangle = m\hbar |j, m\rangle \tag{16}$$

A particular spin system is defined by a particular value of j : for example, spin-1/2 refers to $j = 1/2$. A spin- j system has a basis made up of the eigenvectors of the spin-projection operator, which are indexed by the values of m . Thus, a spin- j system is $(2j + 1)$ -dimensional [17].

The computational basis is labeled by the spins of the subsystems, which makes it convenient if we want to operate on the subsystems independently. However, the composite system has its own total spin and spin-projection, so we can transform to a basis parametrized by these quantities instead. If \mathbf{J}_1^2 and \mathbf{J}_2^2 are the total spin operators for the two subsystems, and J_1^z and J_2^z are the spin-projection operators, then the composite total spin and spin-projection operators are

$$\mathbf{J}^2 = (\mathbf{J}_1 + \mathbf{J}_2)^2, \quad J^z = J_1^z + J_2^z \tag{17}$$

[18]. One can show [18] that for a system composed of a spin- j_1 and a spin- j_2 , the composite total spin has eigenvalues

$$j = j_1 + j_2, j_1 + j_2 - 1, \dots, |j_1 - j_2| \tag{18}$$

with the corresponding possible values of m .

We are now in a position to define the CG transform (for $d = 2$) from a physics perspective:

Definition 3 *The Clebsch-Gordan transform on a system of two spins maps the computational basis to the basis composed of eigenvectors of the composite spin operators.*

There are a number of ways to calculate the CG transform classically (for example, [19] and [20]). A simple method is to begin with the (single) computational basis state with the highest value of the composite total spin and spin projection, and apply the composite lowering operator repeatedly to obtain all of the states with that total spin, then select the next highest composite total spin and spin projection, and so forth.

For $d > 2$, generalizations of the above calculation exist (see [21] for a nice example). For our present purpose it is sufficient to know that the CG transforms for arbitrary d are classically and efficiently calculable, since in our algorithm that part of the computation will be classical.

2.3 Putting the pieces together

We would like to reconcile the two descriptions we have given for the CG transform in the last two sections, starting with the case $d = 2$. We begin by noting the immediate similarities:

1. In the mathematical description of the CG transform, we specialized to decomposing a tensor product of \mathcal{Q}_μ^d (for some $\mu \vdash n$) and $\mathcal{Q}_{(1)}^d$ into irreps. We can think of each of the irreps \mathcal{Q}_μ^d and $\mathcal{Q}_{(1)}^d$ as subsystems to be combined into a larger composite system.
2. The physical description of the CG transform is also based on the concept of combining two subsystems into a larger composite system, with the only qualitative differences being that both of the subsystems are assumed to be definite spins, and that the spins are allowed to be different (recall that in the mathematical description we assumed that all of the boxes had entries in $1, 2, \dots, d$ for the semistandard tableaux).

As it turns out, the generality of the physical CG transform in the dimensions of the subsystems and the generality of the mathematical CG transform in the first irrep in the tensor product are intimately related. This is a consequence of the powerful fact that from a quantum informational perspective, two systems with the same dimension are equivalent as long as they carry the same representation of \mathcal{U}_d (in the current case, \mathcal{U}_2). We take advantage of this fact whenever we talk about a qudit in the abstract: the structure is the same whether the physical qudit is an ion, a photon, or a superconducting quantum circuit. In the $d = 2$ case, the equivalence is between an irrep (in the mathematical picture) and a spin (in the physical picture). Since both are perfectly legitimate quantum systems in their own rights, as long as they have the same dimensions and carry the same representation of \mathcal{U}_2 they can be treated as equivalent.

The operation we will want to perform with our CG transforms is the addition of a single qudit to a larger register of qudits, which is why we constrained the second irrep in our mathematical description to be $\mathcal{Q}_{(1)}^d$. The other qudits in the register will have previously been decomposed into their irreps by a Schur transform, so we can consider their irreps separately: hence the generality of the partition μ that labels the first irrep in our mathematical description (13). Any particular \mathcal{Q}_μ^d is just a Hilbert space whose dimension is determined by the number of semistandard μ -tableaux with entries in $1, 2, \dots, d$ (as given in (9)). This is why, for example, it is correct to say that a composite system of two spin-1/2 particles

decomposes into a spin-1 and a spin-0: the three spin-1 states correspond to the irrep $\mathcal{Q}_{(2)}^2$, and the singlet state, with spin-0, corresponds to the irrep $\mathcal{Q}_{(1,1)}^2$ (see §2.3).

Triplets (spin-1) = $\mathcal{Q}_{(2)}^2$:

$$\left\{ \begin{array}{l} |m=1\rangle = |\frac{1}{2}, \frac{1}{2}\rangle \\ |m=0\rangle = \frac{1}{\sqrt{2}} (|\frac{1}{2}, -\frac{1}{2}\rangle + |-\frac{1}{2}, \frac{1}{2}\rangle) \\ |m=-1\rangle = |-\frac{1}{2}, -\frac{1}{2}\rangle \end{array} \right\} \iff \left\{ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 2 \\ \hline 2 & 2 \\ \hline \end{array} \right\}$$

Singlet (spin-0) = $\mathcal{Q}_{(1,1)}^2$:

$$\left\{ |m=0\rangle = \frac{1}{\sqrt{2}} (|\frac{1}{2}, -\frac{1}{2}\rangle - |-\frac{1}{2}, \frac{1}{2}\rangle) \right\} \iff \left\{ \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} \right\}$$

Fig. 2. CG (Schur) transform on two qubits.

We should not interpret the correspondences in Fig. 2 as direct identifications between the state vectors and tableaux, but the spanned subspaces are correct. The main point is that we can now perform, for example, the CG transform on $\mathcal{Q}_{(2)}^2 \otimes \mathcal{Q}_{(1)}^2$ (i.e., add another qubit), by treating the vector space $\mathcal{Q}_{(2)}^2$ as if it were a spin-1 particle and using the physicists' CG transform. Thus, as we will discuss in the next section, we can use a direct sum of physicists' CG transforms, each of which acts on two systems, to perform a “super-CG transform” that adds one qubit to a register of qubits that have already been decomposed into irreps: this will be the recursion step in our implementation of the Schur transform.

3 Implementation for qubits

In this section we will describe a quantum algorithm that performs the Schur transform on n qubits. We will then prove that this algorithm is efficient: in particular, we will show that it decomposes the Schur transform on n qubits into a sequence of

$$O(n^3) \tag{19}$$

two-level gates. In fact, the number of two-level gates is bounded by $2n^3$; an even tighter upper bound is given in (34). A two-level gate is unitary operator that only acts on two dimensions, i.e., is isomorphic to a 2×2 unitary. Each two-level gate can be decomposed to accuracy ϵ using known methods [22, 23, 24] into $O(n \log(1/\epsilon))$ gates from the Clifford+T set, which can be implemented fault-tolerantly [2, 3].

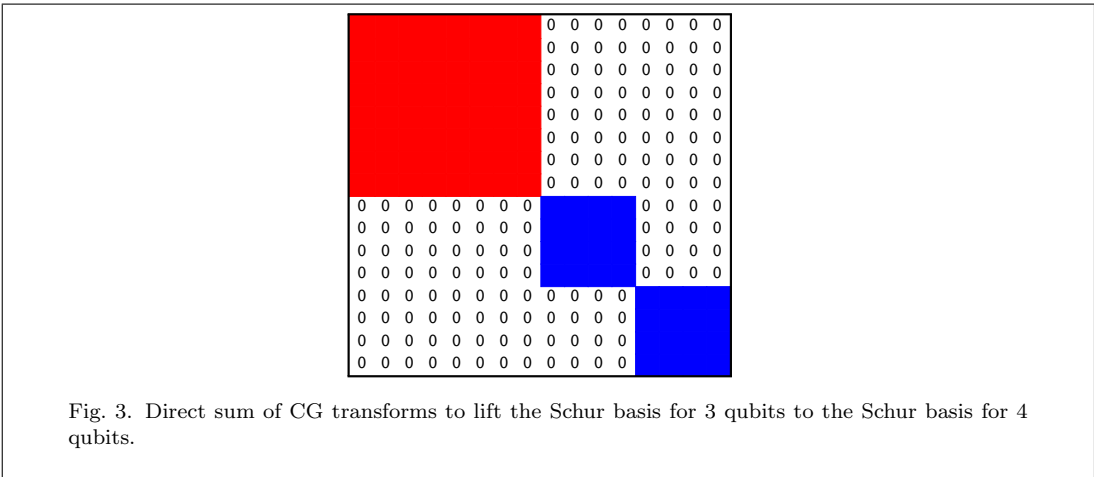
3.1 Recursion structure

Our algorithm is recursive in its outermost layer of structure: this is the element that is shared with BCH's construction [1]. The iteration step looks like this:

- Suppose you are given a basis for the Hilbert space of k qubits that is composed of

eigenvectors of the total spin and spin projection operators for the *whole* system. (We refer to these operators as the *global spin operators*.)

- Then the basis can be broken up into a disjoint union of the bases for several subsystems, each of which has a definite value for the global total spin. For example, if we have $k = 3$ qubits, the Hilbert space decomposes into a spin-3/2 subspace and two spin-1/2 subspaces.
- In other words, assume that we are already in the Schur basis for k qubits.
- Then in order to lift our Schur basis for k qubits to the Schur basis for $k+1$ qubits, all we need to do is apply the appropriate CG transform to each spin-subspace. For example, if we begin with $k = 3$ qubits, to add a fourth qubit, we apply the CG transform that adds a spin-1/2 to a spin-3/2 to the first subspace, which outputs a spin-2 subspace and a spin-1 subspace. We also apply the CG transform that adds a spin-1/2 to a spin-1/2 to each of the spin-1/2 subspaces we already have: each of these outputs a spin-1 subspace and a spin-0 subspace. So in total we end up with a spin-2 subspace, 3 spin-1 subspaces, and 2 spin-0 subspaces: the Schur basis for 4 qubits.
- Thus we can iterate from the Schur basis for k qubits to the Schur basis for $k + 1$ qubits by applying something like a direct sum of CG transforms: see Fig. 3. Continuing the above example, the red block in Fig. 3. corresponds to the CG transform that adds a spin-1/2 to a spin-3/2, and the two blue blocks each correspond to the CG transform that adds a spin-1/2 to a spin-1/2. Fig. 3. does *not* represent the final form of the iteration operator, which we will describe in the next section, but it is a good picture to start from.



- Since the Schur basis for 1 qubit is identical to the computational basis, we can repeatedly apply our iteration operator to obtain the Schur transform on any number of qubits.

- The key to our algorithm is structuring the iteration operator so that it can be implemented efficiently.

3.2 Ordering the basis

We make careful choices of orderings for the bases that appear at each step in our algorithm. By adding a logarithmic (in n) number of ancillary qubits, we can index the various levels of structure in the Schur basis: the multiplicities of the irreps (spin-subspaces), the inequivalent irreps themselves, and finally the states within the irreps. The ordering we will construct is motivated by the observation that most of the inequivalent irreps in the Schur basis have multiplicities greater than one. Therefore, when we perform the iteration step described above, all the copies of each distinct irrep will have a new qubit added to them via the same CG transform. Our ordering allows us to implement each of these distinct CG transforms only once, in parallel over the irreps to which they need to be applied.

We will begin by describing in general the ordering of the Schur basis at any step in the iteration. This will make our description of the iteration step itself straightforward. After any step in the iteration, the Schur basis is encoded in a larger number of qubits than n , so there will be “ghost” states that are not used in the encoding. The encoding qubits are divided into three categories; during the iteration step these categories will become fluid, but between steps they are well-defined. The categories are called:

1. **seq**: qubits that encode the multiplicities of each irrep
2. **par**: qubits that label the inequivalent irreps
3. **stat**: qubits that encode the states within the irreps

We will generally adhere to this order when thinking about tensor products of the qubits. Thus, if we think of the **seq**, **par**, and **stat** qubits as subsystems with some dimensions determined by the numbers of qubits in each, we obtain vectors with the following form:

$$|\mathbf{seq}\rangle \otimes |\mathbf{par}\rangle \otimes |\mathbf{stat}\rangle \quad (20)$$

We note that a similar analysis of these registers, as used in the BCH algorithm, was performed in [14].

A particular state in the Schur basis is encoded in the following way: if the state is the k th state in the i th copy of the j th inequivalent irrep, then in the form (20), it is labeled $|i\rangle \otimes |j\rangle \otimes |k\rangle$. For example, suppose $n = 3$ at the current step, and suppose we want to label the second state in the first spin-1/2 subspace. Indexing from 1, “second state” translates to **stat** = 2, “spin-1/2 subspace” translates to **par** = 2 (assuming we put the spin-3/2 subspace first in **par**), and “first (spin-1/2 subspace)” translates to **seq** = 1. Thus this state is labeled $|1\rangle \otimes |2\rangle \otimes |2\rangle$.

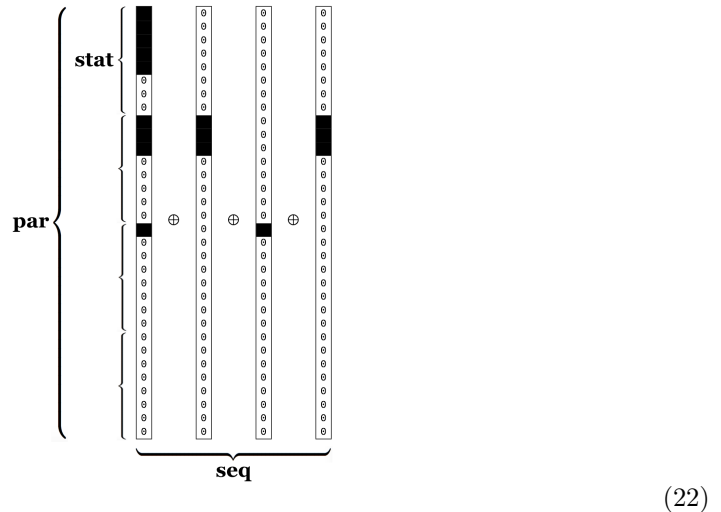
One complication is that within **seq**, the encodings of the multiplicities will not always be ordered in the most intuitive way; but they will be ordered in a calculable way. We will see shortly that this is a consequence of the structure of the iteration step. Before we discuss the structure of the iteration, though, let us consider an example of the basis ordering.

Example:

Suppose $n = 4$. Then we have three inequivalent irreps, corresponding to the three partitions $\lambda \vdash 4$ with degree 2: $\lambda = (4, 0)$, $\lambda = (3, 1)$, and $\lambda = (2, 2)$ (see corollary 1; here we relax our definition of degree-2 to include $(4, 0)$ as a degree-2 partition). To get the multiplicities and dimensions of the irreps, we use (9) and (11). The dimensions (9) are easy to evaluate: the number of degree-2 semistandard tableaux with entries in $\{1, 2\}$ is given by the number of possible locations for the first 2 in the first row, which is $\lambda_1 - \lambda_2 + 1 = n + 1 - 2\lambda_2$. (The multiplicities (11) can be calculated by the hook formula [25], although this is not necessary for the actual running of our algorithm.) We obtain:

$$\begin{aligned}
 \mathcal{Q}_{(4,0)}^2 &: \text{ multiplicity} = 1, \quad \text{dimension} = 5 \\
 \mathcal{Q}_{(3,1)}^2 &: \text{ multiplicity} = 3, \quad \text{dimension} = 3 \\
 \mathcal{Q}_{(2,2)}^2 &: \text{ multiplicity} = 2, \quad \text{dimension} = 1
 \end{aligned}
 \tag{21}$$

(Physically, these irreps correspond to spin-2, spin-1, and spin-0 subspaces, respectively.) We can represent one possible ordering for these irreps schematically by



Here the solid black entries mark the locations of the encoded irreps, and the zeroes mark ghost entries that are not used in the encoding. The largest dimension is 5, so we need 3 qubits (8 states) in **stat** to encode the states within the irreps. There are 3 inequivalent irreps, so we need 2 qubits (4 states) in **par** to identify the inequivalent irreps. The largest multiplicity is 3, so we need 2 qubits (4 states) in **seq** to index the copies of the irreps. Thus in (22), the states of **seq** index the columns. Within each column, there are four “slots,” each comprising 8 states. The slots are indexed by the states of **par**, and each inequivalent irrep is assigned to a particular slot: for example, the copies of $\mathcal{Q}_{(3,1)}^2$ (3-dimensional) all appear in the second slot in their column. The states of **stat** index the states of the irreps within each slot. We will continue to use the term *slot* to refer to the set of states used to encode a particular irrep, and the term *column* to refer to a set of slots used to encode all of the inequivalent irreps.

So, for example, to find the second state in the second copy of $\mathcal{Q}_{(3,1)}^2$, we first find the appropriate column for the second copy of $\mathcal{Q}_{(3,1)}^2$. We then find the second slot within this

column, since $\mathcal{Q}_{(3,1)}^2$ is the second inequivalent irrep: in this case, the second slot is entries 9 through 16 (indexing from 1). We then find the second state within this slot. So in (22), the second state in the second copy of $\mathcal{Q}_{(3,1)}^2$ is encoded in the 10th entry of the second column, or in the notation of (20), $|2\rangle \otimes |2\rangle \otimes |2\rangle$.

The advantage of the ordering we have just described is that the iteration step has the same action on each of the states of **seq** (the columns in (22)). Thus the complexity of the iteration step will be independent of the dimension of **seq**, which is determined by the multiplicities of the irreps. We will see that the exponential growth (with n) in the dimension of the overall Hilbert space is absorbed into the dimension of **seq**, with the dimensions of **par** and **stat** growing only as polynomials in n , so since the complexity of the iteration step is independent of the dimension of **seq**, we obtain polynomial complexity in n for our algorithm.

3.3 Iteration step

We now describe the iteration step that takes a Schur basis of n qubits (as described in the previous section) and lifts it to a Schur basis of $n + 1$ qubits. The iteration step has three pieces:

1. Add the new computational qubit, as well as ancillary qubits if necessary.
2. Perform CG transforms to lift to the new Schur basis. We refer to this step as the “super-CG transform,” since it is a direct sum of CG transforms.
3. Reorder according to the new Schur basis. We refer to this step as the “reordering transform.”

Let us expand these substeps:

1. Add the new computational qubit to the bottom of the register (the end of the tensor product). We can think of adding the new computational qubit as doubling the dimensions of the irrep slots discussed in the previous section. We will see in step 3 that whenever

$$\lfloor \log_2(n+1) \rfloor \neq \lfloor \log_2(n) \rfloor \tag{23}$$

we have to add an additional pair of ancillary qubits in order to properly reorder the Schur basis. Since this happens only “logarithmically often” in n , the total number of ancillary qubits thus added will be logarithmic in n .

2. As we discussed in §3.2, each state of **seq** labels a column composed of slots corresponding to each of the inequivalent irreps \mathcal{Q}_μ^2 . A CG transform is associated to each slot: the CG transform decomposes the composite system of \mathcal{Q}_μ^2 and a new qubit (that is, $\mathcal{Q}_\mu^2 \otimes \mathcal{Q}_{(1)}^2$) into a direct sum of irreps, as given in (13). So the iteration step acts identically on each column, and its action is a direct sum of CG transforms.

3. After performing the CG transforms, the basis is composed of irreps for $n + 1$ qubits, but it is not yet ordered according to the scheme described in §3.2, so the final step is to perform the reordering. The nature of the reordering depends on whether (23) holds. This condition comes from counting the number of qubits required to encode **par** and **stat**:

The number of qubits in **par** is determined by the number of inequivalent irreps, which is equal to the number of degree-2 partitions of n (hence the name “par,” for “partition”). For

$\lambda = (\lambda_1, \lambda_2) \vdash n$, λ_2 can take any value from 0 to $\lfloor n/2 \rfloor$, so the number of degree-2 partitions of n is $\lfloor n/2 \rfloor + 1$. Thus the number of qubits in **par** is

$$|\mathbf{par}| = \lceil \log_2 (\lfloor n/2 \rfloor + 1) \rceil \tag{24}$$

When $n \rightarrow n + 1$, the value of (24) increases by 1 if and only if (23) is true.

The number of qubits in **stat** is determined by the highest dimension of any irrep (“stat” stands for “state”). These dimensions are given by (9), so prior to adding the new qubit, the largest such dimension is associated to the partition $(n, 0)$: $\mathcal{Q}_{(n,0)}^2$ has dimension $n + 1$. Thus the number of qubits in **stat** is

$$|\mathbf{stat}| = \lceil \log_2(n + 1) \rceil \tag{25}$$

When $n \rightarrow n + 1$, the value of (25) also increases by 1 if and only if (23) is true.

The number of qubits in **seq** must be sufficient to encode the highest multiplicity of any irrep. The multiplicity of \mathcal{Q}_λ^2 is given by the number of standard λ -tableaux. This number can be calculated exactly, but we will use a simpler approximation. We can imagine building any standard λ -tableau by inserting the integers 1 to n one at a time in order into the boxes in the Young diagram of λ . Each insertion of an integer must be into the leftmost open box in one of the two rows in the Young diagram, so we have at most two choices for where to place each integer. Further, when we insert 1 we have no choice, since it must be in the upper left box in the diagram, and when we insert n we have no choice, since it must fill the one remaining open box in the diagram. So for *any* $\lambda \vdash n$, we make a sequence of at most $n - 2$ binary choices to generate any standard λ -tableau (hence the name “seq,” for “sequence”), and thus the number of standard λ -tableaux is bounded by 2^{n-2} . Therefore, we take the number of qubits in **seq** to be

$$|\mathbf{seq}| = \log_2 (2^{n-2}) = n - 2 \tag{26}$$

Since the total number of encoding qubits must be at least n , at least one of $|\mathbf{seq}|$, $|\mathbf{par}|$, and $|\mathbf{stat}|$ must scale linearly with n , so we know that (26) is not a drastic overestimate.

Thus, we see that at each iteration, the new computational qubit ends up as a **seq** qubit. Every time (23) is true, we increase $|\mathbf{par}|$ and $|\mathbf{stat}|$ by 1 as well: these are the ancillary qubits. In other words, on every iteration, we double the number of columns. In particular, on an iteration for which (23) is false, the reordering takes each column (whose length has been doubled by the addition of the new computational qubit) and splits it into two columns, each of which has the same length as the original column. On an iteration for which (23) is true, the reordering step requires two additional qubits: each column still splits into two columns, but each of these is 4 times as long as the original column, reflecting the fact that the number of slots and the sizes of the slots have both doubled.

Summing (24), (25), and (26) gives us the total number of qubits used: after some simplification we obtain

$$\text{total number of qubits (exact space requirement)} = n + 2\lceil \log_2(n) \rceil - 1, \tag{27}$$

i.e., the computation requires only $2\lceil \log_2(n) \rceil - 1$ ancillary qubits.

Example:

Consider the iteration $n = 4 \rightarrow 5$. (23) is false, so we expect to only add the new computational qubit. Our input state has the form (22). We show the three pieces of the iteration step for the first column only, since the action will be the same on the other columns, just with fewer active slots:

(28)

In the first step, we add the new computational qubit, doubling the dimensions of the slots. In the second step, we apply the appropriate CG transform to each slot, which splits the tensor product of each original irrep and the new qubit into two new irreps. We have labeled the new irreps by color:

$$\begin{aligned}
 \mathcal{Q}_{(5,0)}^2 &: \text{red in (28)} \\
 \mathcal{Q}_{(4,1)}^2 &: \text{blue in (28)} \\
 \mathcal{Q}_{(3,2)}^2 &: \text{green in (28)}
 \end{aligned}
 \tag{29}$$

In terms of spins, our original irreps were spin-2, spin-1, and spin-0. We can see in (28) that the CG step splits the spin-2 into a spin-(5/2) ($\mathcal{Q}_{(5,0)}^2$) and a spin-(3/2) ($\mathcal{Q}_{(4,1)}^2$), splits the spin-1 into a spin-(3/2) and a spin-(1/2) ($\mathcal{Q}_{(3,2)}^2$), and simply lifts the spin-0 to a spin-(1/2). We then reorder the basis according to these new spin-subspaces. Notice that since there are still only three inequivalent irreps, and the highest dimension is now 6 instead of 5, we still need only 2 **par** qubits (4 slots) and 3 **stat** qubits (8 states within each slot); thus, as we expected, the columns remain the same size.

By repeatedly applying the iteration step described above, we obtain the Schur transform via a product of super-CG and reordering transforms. Each super-CG and reordering transform pair is tensored into the identity matrix of appropriate dimension to copy it over the states of **seq**. We complete our algorithm by decomposing each super-CG transform and reordering transform directly into a product of Clifford+T gates, using known methods for general unitary decomposition [22, 23, 24]. We will discuss this decomposition in more detail in the next section, in which we determine the resulting sequence length.

4 Analysis for qubits

The purpose of the ordering of the basis discussed in §3.2 and §3.3 is to allow the action of the iteration step to be copied over the columns, that is, over the states of **seq**. We now show

how this allows our algorithm for qubits to be efficient.

Our algorithm decomposes the Schur transform on n qubits into $O(n^3)$ two-level unitary operators (unitaries that only act nontrivially on two dimensions). Each of these can be approximated by a sequence of operators from the Clifford+T set [2, 3], which can be implemented fault-tolerantly. We break down the number of two-level unitaries required to construct our algorithm as follows:

- The Schur transform on n qubits requires $n - 1$ iteration steps, since the Schur basis is identical to the computational basis for 1 qubit.
- For the iteration step that lifts $k \rightarrow k + 1$ qubits, the super-CG transform is block diagonal, with each block a CG transform corresponding to the addition of 1 qubit to each inequivalent irrep for k qubits. In particular, for $\mu = (\mu_1, \mu_2) \vdash k$, the block associated to the addition of 1 qubit to \mathcal{Q}_μ^2 has side length equal to twice the dimension of \mathcal{Q}_μ^2 . Thus by (9), the side length of the block corresponding to \mathcal{Q}_μ^2 is

$$2(\mu_1 - \mu_2 + 1) \tag{30}$$

- We can reduce a nonsingular square matrix to upper-triangular using one two-level rotation per nonzero entry below the main diagonal [26]. If we reduce a unitary matrix to upper-triangular, we must have reduced it to a diagonal matrix whose diagonal entries have norm 1, since this is the form of any upper-triangular unitary matrix. By then applying a one-level phase shift (which we are free to think of as a two-level rotation) to each main diagonal entry, we can map that diagonal matrix to the identity matrix. We refer to the total number of two-level rotations into which the goal matrix is decomposed as the *two-level gate sequence length*.
- The number of nonzero entries on or below the main diagonal thus gives us our two-level gate sequence length for a single CG transform. This sum is bounded by the total number of nonzero elements in the CG transform (and is of the same order). Each row in the CG transform is an eigenvector of the total spin projection: suppose a particular row $\langle \phi |$ has total spin projection m . Then for any computational basis vector $|m_1, m_2\rangle$ appearing in the linear combination that forms $|\phi\rangle$, $m_1 + m_2 = m$. But since our CG transforms always just add a single qubit to some irrep \mathcal{Q}_μ^2 , m_2 is restricted to the values $\pm\frac{1}{2}$. Therefore, at most two computational basis vectors appear in the linear combination that forms $|\phi\rangle$, i.e., there are at most two nonzero entries per row in the CG transform. Therefore, the two-level gate sequence length for the CG transform is bounded by 2ℓ , where ℓ is the side length of the CG transform.
- In our case, $\ell = 2(\mu_1 - \mu_2 + 1)$, so the sequence length for the CG transform on \mathcal{Q}_μ^2 is bounded by

$$4(\mu_1 - \mu_2 + 1) = 4(k - 2\mu_2 + 1) \tag{31}$$

for $\mu_1 + \mu_2 = k$.

- The super-CG transform is a direct sum of CG transforms on \mathcal{Q}_μ^2 , for every $\mu \vdash k$ with degree 2. The possible μ are

$$\mu \in \left\{ (k - \mu_2, \mu_2) \mid \mu_2 \in \left\{ 0, 1, 2, \dots, \left\lfloor \frac{k}{2} \right\rfloor \right\} \right\} \tag{32}$$

Therefore, the sequence length for the super-CG transform is

$$4 \sum_{\mu_2=0}^{\lfloor k/2 \rfloor} (k - 2\mu_2 + 1) \quad (33)$$

- Our Schur transform decomposes into super-CG transforms and reordering transforms for $k = 1, 2, \dots, n - 1$, as discussed above; so far we have only discussed the sequence lengths for the super-CG transforms. Each reordering transform is a permutation matrix that permutes exactly the same entries that its corresponding super-CG transform acts on. Thus, it has at most 1 off-diagonal element for each of these entries. Since the super-CG transform had at most 2, the effect of including the reordering transform in our analysis is simply to increase the constant factor in (33) from 4 to 6. Thus, the two-level gate sequence length for the full Schur transform is bounded by

$$6 \sum_{k=1}^{n-1} \sum_{\mu_2=0}^{\lfloor k/2 \rfloor} (k - 2\mu_2 + 1) \leq \frac{1}{3}n^3 + \frac{9}{4}n^2 + \frac{13}{4}n - 6 < 2n^3 \quad (34)$$

for $n \geq 2$; this is $O(n^3)$, as desired.

We can test our prediction for the two-level gate sequence length directly. In §4, the black line is n^3 , and the solid points are the actual two-level gate sequence lengths generated by our algorithm for the Schur transform. As it turns out, the actual two-level gate sequence length appears to be bounded by n^3 ; the additional factor of 2 in (34) arises as a result of allowances we made for the sake of simplicity in our analysis.

It also behooves us at this point to note that one does not obtain efficient sequence lengths by direct decomposition of the Schur transform into two-level gates using a method such as that described in [22], i.e., by bypassing our algorithm entirely. We expect the sequence lengths thus generated to be exponential in scaling, and this is indeed what we obtain: the open circles in §4 are the two-level gate sequence lengths generated by decomposing the Schur transform directly.

Now that we have our sequence of $O(n^3)$ two-level rotations, we can approximate each two-level rotation by Clifford+T operators using known methods [22, 23, 24]: a two-level rotation on n qubits can be decomposed into $O(n \log(1/\delta))$ Clifford+T operators, with error δ . If we want to achieve an overall error bounded by ϵ , we can calculate the error bound required for the individual two-level rotations as follows:

Lemma 1 *Let A_1, A_2, \dots, A_m be a sequence of unitary operators. If B_1, B_2, \dots, B_m is a sequence of unitary operators such that B_j approximates A_j with error δ in the trace norm for all j , then the product $\Pi_{j=1}^m B_j$ approximates the product $\Pi_{j=1}^m A_j$ with error $\epsilon \leq m\delta$ [28].*

Therefore, if we want our overall error to be bounded by ϵ , the errors for our decomposition matrices (the two-level rotations) must be bounded by $\delta = \epsilon/m$. We decompose into $m = O(n^3)$ two-level rotations, so our decomposition errors for the individual rotations must be bounded by $\frac{\epsilon}{an^3}$ for some scalar a . Thus the lengths of the approximating sequences will be bounded by

$$O\left(n \log\left(\frac{an^3}{\epsilon}\right)\right) = O\left(3n \log\left(\frac{n}{\epsilon}\right)\right) = O\left(n \log\left(\frac{n}{\epsilon}\right)\right) \quad (35)$$

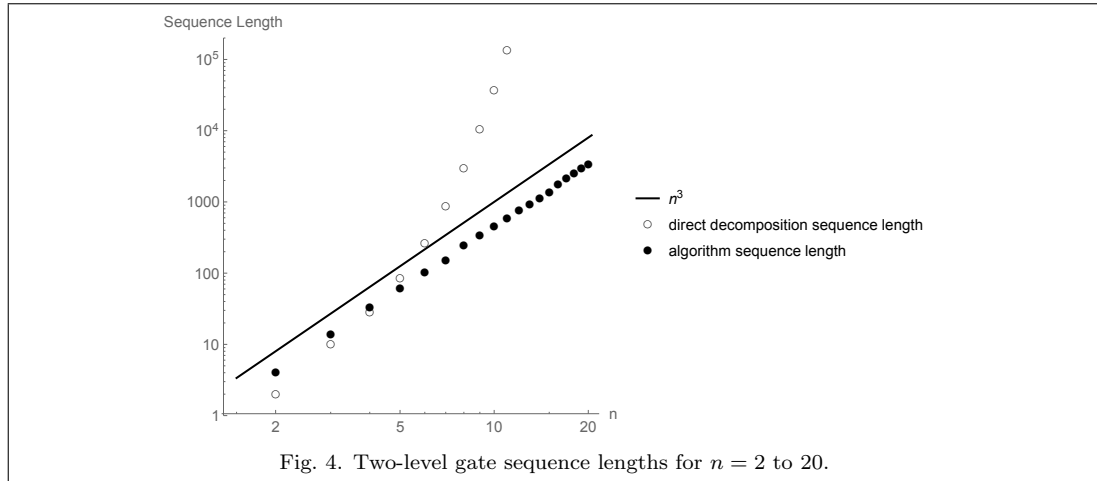


Fig. 4. Two-level gate sequence lengths for $n = 2$ to 20.

Multiplying by the number of two-level gates in our decomposition gives us the overall sequence length for decomposition of the Schur transform on n qubits into Clifford+T operators:

$$O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right) \tag{36}$$

We show in §6 that this length agrees with that of the circuit schematically described in [4].

We have implemented this method of calculating an efficient quantum algorithm for the qubit Schur transform both as a Python script and in Mathematica: the code is available online at [29]. We used our implementation to generate the sequence lengths in Fig. 4, and also tested it for correctness by multiplying out the resulting Schur transforms and checking that

1. they match previous calculations,
2. they diagonalize the global spin operators, and
3. they reduce the representations of \mathcal{U}_2 and \mathcal{S}_n described in §1.1 to direct sums of irreps.

5 Implementation and analysis for qudits

We now generalize from qubits to qudits of dimension d . Up to and including the decomposition into two-level gates, the only differences will be in the dimensions, numbers, and multiplicities of the irreps, as well as in the CG transforms used. The similarity ends with the decomposition of the two-level rotations into primitives from a universal gate set. The Clifford+T gate set is specific to qubits, so in order to complete this piece of the construction, we need a different universal gate set that applies to general qudits. Ideally, this gate set would be fault-tolerant, and the decomposition of a two-level rotation into primitives from the gate set would be $O(n \log(1/\epsilon))$. To our knowledge, no constructive algorithm that satisfies these conditions has been found. However, Gottesman showed that fault-tolerant computation with qudits is possible, and provided a generalization of the Clifford+T set that is universal and fault-tolerant [30]. Thus, we can apply the Solovay-Kitaev algorithm [31, 32] to this set to obtain a fault-tolerant decomposition of any two-level rotation into $O(n \log^p(1/\epsilon))$ primitives,

for $p \approx 3.97$. This will be sufficient to show that our construction gives a efficient decomposition of the Schur transform into primitives; the only further improvements would be in the dependence on ϵ , for which p could in principle be decreased to 1 (note: this has been done [33], but not fault-tolerantly).

We can break down the number of two-level rotations required to construct our algorithm for the Schur transform on n qudits (dimension d) as follows:

- The Schur transform on n qudits requires $n - 1$ super-CG transforms.
- The super-CG transform that adds 1 qudit to k qudits is block diagonal, with each block corresponding to the addition of 1 qudit to an irrep of k qudits. In particular, for $\mu = (\mu_1, \mu_2, \dots, \mu_d) \vdash k$, the block associated to the addition of 1 qudit to \mathcal{Q}_μ^d has side length equal to d times the dimension of \mathcal{Q}_μ^d . Note that μ cannot have degree greater than d , so we can write $\mu = (\mu_1, \mu_2, \dots, \mu_d)$ as long as we remember that μ_2 through μ_d may be 0.
- The dimension of \mathcal{Q}_μ^d is given by the number of semistandard μ -tableaux t with entries in $\{1, 2, \dots, d\}$ (9). For $\mu \vdash k$, μ has the maximal number of semistandard tableaux if $\mu = (k)$, as was the case for qubits. The number of semistandard tableaux with shape (k) and entries in $\{1, 2, \dots, d\}$ is bounded by

$$(k + 1)^{d-1} \quad (37)$$

since we choose the location of the first 2, the first 3, ... , and the first d in the tableau, each out of at most $k + 1$ possible locations (including one outside the tableau).

- The number of distinct irreps is given by the number of distinct partitions of k with order bounded by d . This number is bounded by k^d , since we choose the length of each of the d rows from at most k possibilities.
- Our super-CG transform on k qudits will have a block corresponding to each distinct irrep of k qudits. The side length of each of these blocks is d times the dimension of the corresponding irrep, so the number of two-level rotations required to implement the block is bounded by the square of this side length. Since the dimension of any irrep is bounded by $(k + 1)^{d-1}$ for k qudits, the side length of the corresponding block is bounded by $d(k + 1)^{d-1}$. Thus, since the number of distinct irreps is bounded by k^d , and hence also by $(k + 1)^d$, the number of nonzero entries in the super-CG is bounded by

$$d^2(k + 1)^{2d-2} \cdot (k + 1)^d = d^2(k + 1)^{3d-2} \quad (38)$$

This is a bound on the number of two-level rotations to decompose the super-CG transform.

- To build the Schur transform, we take the product of super-CG transforms on k qudits for k from 1 to $n - 1$, so the total number of two-level rotations to decompose the Schur transform on n qudits is bounded (for fixed d) by

$$d^2 \sum_{k=1}^{n-1} (k + 1)^{3d-2} \leq d^2 \int_{k=1}^n (k + 1)^{3d-2} dk < d(n + 1)^{3d-1} < O(dn^{3d-1}) \quad (39)$$

As discussed above, each two-level rotation can be decomposed to accuracy δ into $O(n \log^p(1/\delta))$ fault-tolerant primitives (for $p \approx 3.97$), so the Schur transform can be decomposed into a product of fault-tolerant primitives whose length is

$$O\left(d n^{3d} \log^p\left(\frac{dn^{3d-1}}{\epsilon}\right)\right) = O\left(d^{1+p} n^{3d} \log^p\left(\frac{dn}{\epsilon}\right)\right) \tag{40}$$

for overall error bounded by ϵ . Since this is polynomial in n , our goal is achieved.

A tighter analysis than we have given is certainly possible, but ours is sufficient for proof of principle.

6 Discussion

It is appropriate for us to conclude by comparing the algorithms presented here with those of BCH [1, 4] and summarizing what we have contributed to the study of the quantum Schur transform. The following is an itemization of the similarities and differences between these algorithms:

1. BCH's algorithm and ours each implement the Schur transform on n qubits or qudits.
2. BCH's algorithm and ours share the same recursive structure at the outermost level, but ours uses a different structure for qudits at lower levels (expanding upon arguments given in Section V in [1]). In this sense we have extended the BCH algorithm.
3. Our algorithm has low constant overhead in terms of the decomposition into two-level gates: for qubits this sequence length is bounded by $2n^3$, and for qudits it is bounded by $d(n+1)^{3d-1}$.
4. For the case of qubits, the asymptotic sequence lengths of these algorithms, in terms of Clifford+T gates, are identical. Specifically, we can analyze the BCH sequence length using Eq. (7) in [4], in which the CG transform is written as a rotation controlled on states $|j, m\rangle$. m takes $2j+1$ possible values for each value of j . Each iteration of the CG transform is associated to some maximum value J of j , so each CG transform has $\sum_{j \leq J} (2j+1)$ distinct controls (assuming some optimal method of bookkeeping, such as we propose, to copy operations over identical values of j). Summing over the values of J (from $1/2$ to $n/2$ in steps of $1/2$), gives a total of $O(n^3)$ controls for the Schur transform. Decomposing these into Clifford+T gates (or any universal set of gates that act on a constant number of qubits), for an overall error bounded by ϵ , we obtain a total sequence length of $O(n^4 \text{ poly } \log(n/\epsilon))$, which matches our qubit sequence length (36) in its dependence on n .
5. Our algorithm employs exactly $2\lceil \log_2(n) \rceil - 1$ ancillary qubits. A direct implementation of the BCH algorithm would use n ancillary qubits (although these can be compressed afterwards [1]). In this sense we have simplified the BCH qubit algorithm.
6. For the case of qudits, we provide an explicit asymptotic sequence length that is polynomial in n ; BCH specify that their algorithm is polynomial in n and d but do not state the exponents. However, in many proposed quantum computation architectures, d is a

constant with n the only variable. In this case, the two algorithms are asymptotically identical (in the sense that both are polynomial).

7. The main advantage of BCH's qudit algorithm is that it is polynomial in d , while our qudit algorithm is exponential in d . This comes at the cost of constructing the required Clebsch-Gordan coefficients by using a reduced Wigner operator (via the Wigner-Eckart Theorem).
8. The main advantage of our qudit algorithm is its relative simplicity. Our algorithm includes only the most basic operations required to implement the Schur transform. BCH employ additional mathematical machinery to reduce their sequence length's asymptotic dependence on d to polynomial. Part of our contribution is to show that if we relax that requirement, with the understanding that in many quantum computers the asymptotic dependence on d will be irrelevant, we can obtain an algorithm that is efficient in n using only the most basic mathematical components.
9. These most basic mathematical components are the CG transforms. At each step in the recursive structure of our algorithm (or BCH's), we must implement a CG transform to act on each irrep output from the previous step. The CG transforms acting on equivalent irreps are identical, so the minimum number of operations for the step is to implement each of these once; this is exactly what we accomplish.
10. To complete the Schur transform all we must do is route the correct outputs from one step into the correct inputs from the next. The calculation of CG transforms is a solved problem: CG coefficients could be looked up from a precalculated table if we desired (e.g., using [21]), and doing so would not change our sequence length. Thus, in a real sense we have reduced the Schur transform to a succession of permutations of the basis that implement the "routing." In this sense we have simplified the BCH qudit algorithm.
11. The remainder of our contribution is to provide an explicit implementation of the qubit version of the algorithm, described in §3 and also available online [29], which allows us to verify our results (see Fig. 4). Given the various applications for the Schur transform, it is our hope that this implementation and our presentation will be useful to others approaching quantum computation and the Schur transform from a computer science or physics background.

Acknowledgements

We would like to thank William Wootters for his helpful comments.

References

- [1] D. Bacon, I. L. Chuang, and A. W. Harrow, Proc. 18th Ann. ACM-SIAM SODA pp. 1235–1244 (2007).
- [2] D. Gottesman, Phys. Rev. A **57**, 127 (1998).
- [3] S. Bravyi and A. Kitaev, Phys. Rev. A **71**, 022316 (2005).

- [4] D. Bacon, I. L. Chuang, and A. W. Harrow, *Phys. Rev. Lett.* **97**, 170502 (2006).
- [5] R. Goodman and N. R. Wallach, *Representations and Invariants of the Classical Groups*, vol. 68 of *Encyclopedia of Mathematics and Its Applications* (Cambridge University Press, Cambridge, UK, 1998).
- [6] P. Zanardi and M. Rasetti, *Phys. Rev. Lett.* **79**, 3306 (1997).
- [7] D. A. Lidar, I. L. Chuang, and K. B. Whaley, *Phys. Rev. Lett.* **81**, 2594 (1998).
- [8] C. A. Bishop and M. S. Byrd, *J. Phys. A: Math. Theor.* **42**, 055301 (2009).
- [9] K. Audenaert, M. Nussbaum, A. Szkola, and F. Verstraete, *Commun. Math. Phys.* **279**, 251 (2008).
- [10] M. Hayashi, *J. Phys. A: Math. Theor.* **35**, 10759 (2002).
- [11] M. Keyl and R. F. Werner, *Phys. Rev. A* **64**, 052311 (2001).
- [12] R. D. Gill and S. Massar, *Phys. Rev. A* **61**, 042312 (2002).
- [13] K. Matsumoto and M. Hayashi, *Phys. Rev. A* **75**, 0632338 (2007).
- [14] R. Blume-Kohout, S. Croke, and D. Gottesman, *IEEE Transactions on Information Theory* **60**, 334 (2014).
- [15] S. D. Bartlett, T. Rudolph, and R. W. Spekkens, *Phys. Rev. Lett.* **91**, 027901 (2003).
- [16] S. J. Berg, Ph.D. thesis, University of California (2012).
- [17] J. S. Townsend, *A Modern Approach to Quantum Mechanics* (University Science Books, Mill Valley, California, 2012), 2nd ed.
- [18] J. Sakurai, *Modern Quantum Mechanics* (Addison-Wesley Publishing Company, Inc., USA, 1985).
- [19] K. Schertler and M. H. Thoma, *Annalen Phys.* **5**, 103 (1996).
- [20] P. G. Burke, *Clebsch-Gordan and Racah Coefficients* (Springer International Publishing AG., 2010), vol. 61 of *Springer Series on Atomic, Optical, and Plasma Physics*, chap. R-Matrix Theory of Atomic Collisions, pp. 607–617.
- [21] A. Alex, M. Kalus, A. Huckleberry, and J. von Delft, *J. Math. Phys.* **52**, 023507 (2011).
- [22] V. Kliuchnikov, arXiv: 1306.3200, (2013).
- [23] V. Kliuchnikov, D. Maslov, and M. Mosca, *Phys. Rev. Lett.* **110**, 190502 (2013).
- [24] P. Selinger, *Quantum Information and Computation* **15**, 159 (2015).
- [25] B. E. Sagan, *The Symmetric Group, Representations, Combinatorial Algorithms, and Symmetric Functions*, no. 203 in *Graduate Texts in Mathematics* (Springer-Verlag New York, 2001), 2nd ed.

- [26] D. S. Bernstein, *Matrix Mathematics* (Princeton University Press, 2005).
- [27] W. M. Kirby, *A practical quantum schur transform* (2017).
- [28] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, UK, 2001).
- [29] URL <http://www.github.com/wmkirby1/schur-transform>, (2018).
- [30] D. Gottesman, *Chaos Solitons Fractals* **10**, 1749 (1999).
- [31] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation, volume 47 of Graduate Studies in Mathematics* (American Mathematical Society, Providence, RI, 2002).
- [32] C. M. Dawson and M. A. Nielsen, *Quantum Information and Computation* **0**, 1 (2005).
- [33] D.-S. Wang and B. C. Sanders, *New J. Phys.* **17**, 043004 (2015).