Efficient Ancilla-Free Reversible and Quantum Circuits for the Hidden Weighted Bit Function

Sergey Bravyi, Theodore J. Yoder, and Dmitri Maslov, *Fellow, IEEE*

Abstract—The Hidden Weighted Bit function plays an important role in the study of classical models of computation. A common belief is that this function is exponentially hard to implement using reversible ancilla-free circuits, even though introducing a small number of ancillae allows a very efficient implementation. In this paper, we refute the exponential hardness conjecture by developing a polynomial-size reversible ancilla-free circuit computing the Hidden Weighted Bit function. Our circuit has size $O(n^{6.42})$, where n is the number of input bits. We also show that the Hidden Weighted Bit function can be computed by a quantum ancilla-free circuit of size $O(n^2)$. The technical tools employed come from a combination of Theoretical Computer Science (Barrington's theorem) and Physics (simulation of fermionic Hamiltonians) techniques.

Index Terms—Quantum computing, quantum circuits, reversible circuits

1 Introduction

The origins of the Hidden Weighted Bit function go back to the study of models of classical computation. This function, denoted HWB, takes as input an n-bit string x and outputs the kth bit of x, where k is the Hamming weight of x; if the input weight is 0, the output is 0. It is best known for combining the ease of algorithmic description and implementation by classical Boolean circuits with the hardness of representation by Ordered Binary Decision Diagrams (OBDDs) [1]—a popular tool in VLSI CAD [2]. The difference between complexities of logarithmic-depth implementations of HWB by circuits (recall that $\mathsf{HWB} \in \mathrm{NC}^1$ but $\mathsf{HWB} \notin \mathrm{AC}^0$) and an exponential lower bound for the size of the OBDD [3] is a startling two exponents (logarithm vs exponent). Relaxing the constraints on the type of Binary Decision Diagram considered or restricting the computations by circuits enables a multitude of implementations with polynomial cost [4]. Relevant to this paper, we highlight that removing the constraint that the order of variables in the OBDD is fixed allows implementing HWB as a polynomial-size BDD (equivalently, branching program) [5].

The Hidden Weighted Bit function was first introduced in the context of reversible and quantum computations about 16 years ago by I. L. Markov and K. N. Patel (unpublished), and the earliest explicit mention dates to the year 2005 [6]. The original specification is irreversible, and required a slight modification to comply with the restrictions of reversible and quantum computations. Specifically, the Hidden Weighted Bit function was redefined to become the cyclic shift to the right by the input weight. We denote this reversible specification as **hwb**. Formally, $\mathbf{hwb}(x)$ is defined as the cyclic shift of its input x to the right by W positions, where $W = x_1 + x_2 + \ldots + x_n$ is the Hamming weight of x. The following shows the truth table of 3-input **hwb**:

x	000	100	010	110	001	101	011	111
$\mathbf{hwb}(x)$	000	010	001	101	100	011	110	111

Since its introduction, **hwb** was used by numerous authors focusing on the synthesis and optimization of reversible and quantum circuits as a test case.

Despite a stream of improvements in the respective circuit sizes by various research groups [7], [8], [9], [10], the best known ancilla-free reversible circuits exhibit exponential scaling in the number of gates. The synthesis algorithms benefiting from the inclusion of additional gates, such as multiple-control multiple-target Toffoli, Fredkin, and Peres gates [6], [9], [11] also failed to find an efficient implementation without ancillae. In 2013, this culminated with the **hwb** receiving the designation of a "hard" benchmark function [12]. A recent asymptotically optimal synthesis algorithm over the library with NOT, CNOT, and Toffoli gates [13], introduced in the year 2015, was also unable to find an efficient ancilla-free implementation. An ancilla-free quantum circuit can be obtained by employing an asymptotically optimal quantum circuit synthesis algorithm such as [14], but the quantum gate count appears to remain exponential and larger than what is possible to obtain through the application of the asymptotically optimal reversible logic synthesis algorithm [13].

The introduction of even a small number of ancillae changes the picture dramatically. Just $O(\log(n))$ ancillary (qu) bits suffice to develop a reversible circuit with $O(n\log^2(n))$ gates [15], by following the algorithmic definition of this function. Specifically, the circuit first computes the input weight into a clean ancilla register spanning $\lceil \log(n) \rceil$ bits, then performs control-SWAPs, and finally uncomputes the input

• *The authors are with the IBM Quantum, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA. E-mail: sbravyi@us.ibm.com, {ted.yoder, dmitri.maslov}@ibm.com.*

Manuscript received 21 July 2020; revised 25 Jan. 2021; accepted 25 Apr. 2021. Date of publication 28 Apr. 2021; date of current version 7 Apr. 2022. (Corresponding author: Dmitri Maslov.) Recommended for acceptance by K. Gaj. Digital Object Identifier no. 10.1109/TC.2021.3076435

weight calculation to return ancillae back to zero. A more complex but still polynomial-size circuit is given by Barrington's theorem which offers a width-5 polynomial-size branching program implementation of arbitrary functions in the NC^1 class, including individual components of the input weight. The reversible circuit relies on using only 3 $(=\lceil \log(5) \rceil)$ ancillary bits. It can be obtained by computing the individual bits of the input weight through Barrington's theorem, and using such bits logarithmically many times to control-SWAP the respective input bits into their desired positions. Finally, the existence of a polynomial-size quantum circuit using a single ancilla follows from [16], which invokes a version of Barrington's theorem by packing the ancillary qubit with weight information. With 1 being the smallest non-zero positive integer number, it seemed that the use of at least a single ancilla would be required to compute the weight into before using it to control-SWAP the bits according to the **hwb** specification. Indeed, all other attempts to keep both n input bits and the input weight to a register of only n bits would seem impossible due to the lack of space—the input bits already take up all of it. There is also no other known algorithmic description of the **hwb** function that would allow to compute it without relying on additional bits.

State of the art, in both the classical reversible and quantum settings, thus suggests an exponential difference in the gate count between circuits with no ancillae and circuits with a constant number of ancillae. Thus one of the following two statements has to be true: either it is possible that introducing a single ancilla may reduce the circuit size exponentially, or it is possible to use n bits to store n Boolean values together with their own weight. Each statement, if true, would be uniquely surprising.

In this paper, we demonstrate efficient implementations of the **hwb** function by ancilla-free reversible and quantum circuits, thereby resolving which of the above two statements is correct. Our reversible circuit algorithmically replicates the **hwb** definition using no additional space—in effect, it encodes the weight into the order of bits and thus does not contradict the above intuition that appeared to prohibit carrying both information on the n input bits and the weight over just n bits. Our reversible ancilla-free circuit requires $O(n^{6.42})$ gates and our quantum ancilla-free circuit requires $O(n^2)$ gates. These results furthermore refute the exponential hardness conjecture and remove **hwb** from the class of hard benchmarks [12].

We next sketch the main ideas behind our ancilla-free circuits. We begin with the reversible circuit. Our construction works as follows. First, we show that the n-bit **hwb** function can be decomposed into a product of $O(n\log(n))$ gates denoted $C5(f(x);B)$, where $f(x)$ is a symmetric Boolean function and $B \subset x$ is a subset with 5 input bits. The gate $C5(f;B)$ cyclically shifts the 5-bit register B if $f(x)=1$, and does nothing when $f(x)=0$. To implement $C5(f;B)$, we first break it down into a product of 6 gates of the form $C5|_{M_i}(f(x\backslash B);B)$, where $i \in \{1,2,3,4,5,6\}$, each M_i is a fixed set of Boolean 5-tuples, and f are symmetric Boolean functions. The gate $C5|_{M_i}$ restricts the operation of the corresponding gate $C5$ onto the set M_i and simultaneously separates the set B of bits being cycle-shifted from the set $x\backslash B$ controlling these shifts. This allows us to employ Barrington's theorem [5] to implement the gates $C5|_{M_i}(f(x\backslash B);B)$ in an ancilla-free fashion by expressing them as polynomial-size branching programs with the input $x\backslash B$ and computing into B. Each instruction in such program realizes a permutation of 5-bit strings controlled by a single bit and it can thus be mapped into a reversible circuit over $6=5+1$ wires.

Next we introduce our quantum ancilla-free circuit. Let $U_{\mathbf{hwb}}$ be the n-qubit unitary operator implementing the **hwb** function. By definition, $U_{\mathbf{hwb}}|x\rangle = \mathsf{C}^{x_1+x_2+\ldots+x_n}|x\rangle$, where C is the cyclic shift of n qubits. Suppose we can find an n-qubit Hamiltonian H such that $\mathsf{C}=e^{iH}$ and H commutes with the Hamming weight operator $W=\sum_{j=1}^{n}|1\rangle\langle 1|_j$. Then $U_{\mathbf{hwb}} = e^{iHW}$. Thus it suffices to construct a quantum circuit simulating the time evolution under the Hamiltonian HW. Since the cyclic shift C is analogous to the translation operator for a particle moving on a circle, the Hamiltonian H generating the cyclic shift C is analogous to the particle's momentum operator [17]. This observation suggests that H can be diagonalized by a suitable Fourier transform. We formalize this intuition using the language of fermions and the fermionic Fourier transform, which is routinely used in physics and quantum simulation algorithms [19], [20]. The desired Hamiltonian H such that $\mathsf{C}=e^{iH}$ is shown to have the form $H=V^{\dagger}H'V$, where V is a (modified) fermionic Fourier transform and H' is a simple diagonal Hamiltonian. We also show that V commutes with the Hamming weight operator W, so that $U_{\mathbf{hwb}}=e^{iHW}=V^{\dagger}e^{iH'W}V$. We demonstrate that each layer in this decomposition of $U_{\mathbf{hwb}}$ can be implemented by a quantum circuit of size $O(n^2)$.

The rest of the paper is organized as follows. Section 2 introduces a simple modification of the known $O(n\log^2(n))$-gate $O(\log(n))$-ancilla reversible circuit that requires $O(n\log(n))$ gates and $O(\log(n))$ ancillary bits. Section 3 describes an $O(n^{6.42})$-gate ancilla-free reversible circuit. Section 4 reports an ancilla-free $O(n^2)$-gate quantum circuit. These sections are independent of each other and can be read in any order.

2 Reversible Circuit of Size $O(n\log n)$ Using Ancillae

We start with the description of a modification of the previously reported classical/reversible circuit that implements **hwb** with $O(n\log^2(n))$ gates and about $\log(n)$ ancillae [15]. Our circuit relies on $O(n\log(n))$ gates and about $2\log(n)$ ancillae. Compared to the original, it features improved asymptotics at the cost of using twice the computational/ancillary space.

Similarly to [15], we break down the computation into three stages:

1) Compute the input weight $W=x_1+x_2+\ldots+x_n$.
2) Apply controlled-SWAP gates to SWAP inputs into their correct position as specified by the **hwb**.
3) Restore the value of ancillary register to $|0\rangle$ by appending the inverse of the stage 1.

Note that the stage 3 is omitted in [15], allowing a direct comparison to our circuit illustrated in Fig. 1. The difference between our construction and [15] is how we compute the input weight. Specifically, we use the same "plus-one" approach to calculate the weight into the ancillary register, however, we implement the integer increment function differently. Given input x_i, $1 \le i \le n$, the register $w_1, w_2, \ldots, w_{\lfloor \log(i) \rfloor}$

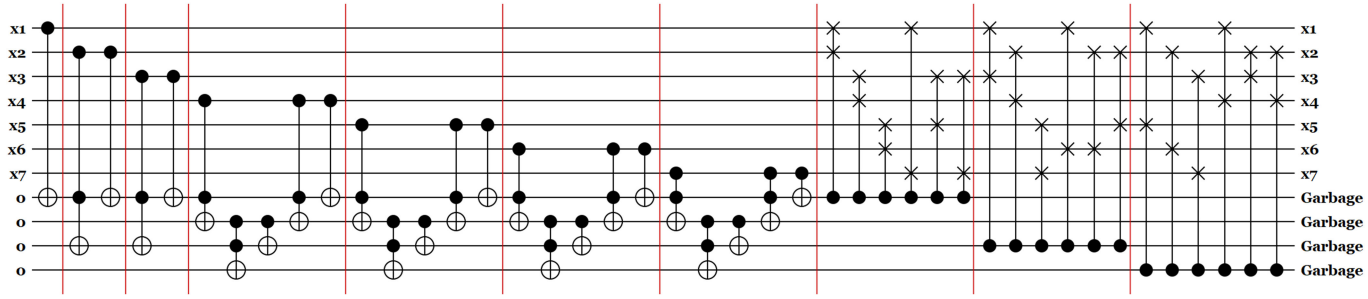

Fig. 1. 10-stage reversible circuit applying the 7-bit **hwb** to $|x_1x_2x_3x_4x_5x_6x_7w_1t_1w_2w_3\rangle$. Each of the first 7 CNOT/TOFFOLI gate stages increments $|w_1w_2w_3\rangle$ by one depending on the value of input variable, the next 3 FREDKIN gate stages perform controlled-SWAP. Vertical red lines separate these 10 stages. Not shown is Garbage uncomputation that can be performed by appending the inversion of the weight calculation circuit (CNOT/TOFFOLI gate part).

$+1$, where the input weight is being computed into, and temporary storage $t_1, t_2, \ldots, t_{\lfloor \log(i) \rfloor - 1}$, "increment by one" works as follows. If $i = 1$, apply CNOT$(x1; w1)$. For $i > 1$:

1) if $i > 3$: apply Toffoli gate to $|x_i, w_1, t_1\rangle$; for j from 2 to $\lfloor \log(i) \rfloor - 1$ apply the Toffoli gate TOFFOLI$(t_{j-1}, w_j; t_j)$;
2) if $i = 2$ or $i = 3$ apply TOFFOLI$(x_j, w_1; w_2)$ CNOT$(x_j; w_1)$; else apply TOFFOLI$(t_{\lfloor \log(i) \rfloor - 1}, w_{\lfloor \log(i) \rfloor}; w_{\lfloor \log(i) \rfloor + 1})$ CNOT$(t_{\lfloor \log(i) \rfloor - 1}; w_{\lfloor \log(i) \rfloor})$.
3) if $i > 3$: for j from $\lfloor \log(i) \rfloor - 1$ down to 2 apply the half adder, computed by the circuit TOFFOLI$(t_{j-1}, w_j; t_j)$CNOT$(t_{j-1}; w_j)$. Apply TOFFOLI$(x_i, w_1; t_1)$CNOT$(x_i; w_1)$.

In our implementation, the t register is used to store necessary digit shifts. Advertised asymptotics follow by inspection of the above construction. We furthermore illustrated our circuit in Fig. 1 for $n = 7$.

3 Ancilla-Free Reversible Circuit of Size $O(n^{6.42})$

In this section we show how to construct an ancilla-free classical reversible circuit of size $poly(n)$ implementing **hwb**. We focus on $n \geq 5$, noting that optimal circuits with n up to 4 are already known.

Let n be the total number of bits, and $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ be the input. In some discussions where it is convenient, we label these bits by the integers $\{0, 1, \ldots, n-1\} = \mathbb{Z}_n$. Suppose $B \subseteq \mathbb{Z}_n$ is a subset of 5 bits and $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function (that is, $f(x)$ depends only on the Hamming weight of x). Define a reversible gate

$$C5(f; B) : \{0,1\}^n \to \{0,1\}^n,$$

where the output is obtained from the input x by applying the cyclic shift to the register B if $f(x)$=1. Otherwise, when $f(x) = 0$, the gate does nothing. Note that, because the symmetric function f does not depend on the order of the bits, $C5(f; B)$ is a permutation of the set $\{0,1\}^n$. Moreover, $C5(f; B)$ is an even permutation, since it is a product of length-5 cycles and each length-5 cycle is an even permutation.

Define $C(f; (i_0, i_1, \ldots, i_{t-1}))$ to be a reversible gate that applies the cyclic shift of some t bits defined by the cycle $(i_0, i_1, \ldots, i_{t-1})$ (where $i_0, i_1, \ldots, i_{t-1} \in \mathbb{Z}_n$ are all distinct) if the symmetric function f evaluates to one and does nothing otherwise. We call $i_0, i_1, \ldots, i_{t-1}$ the targets. We call a collection of C-type gates a layer when the sets of their targets do not overlap.

We next construct **hwb** by first expressing it as a circuit with the C-type gates, then breaking down the C-type gates into elementary reversible gates and $C5$-type gates, and finally expressing the $C5$-type gates in terms of the elementary reversible gates.

Lemma 1. *The n-bit* **hwb** *function can be implemented by an ancilla-free circuit with $\lfloor \log(n) \rfloor + 1$ layers of C-type gates.*

Proof. We will create a circuit with k layers numbered $0, 1, \ldots, \lfloor \log(n) \rfloor$. At each layer, the C gates take the form $C(f_k; *)$. Select the symmetric functions f_k as follows: let $f_k(x) = 1$ iff the kth power of 2 in the binary expansion of the weight $W = x_1 + x_2 + ... + x_n$ equals one. Note that f_k are symmetric functions since the calculation of weight does not depend on the order the bits are added in. The function **hwb** can now be expressed as

$$\textbf{hwb} = C^{2^0}(f_0; (0, 1, \ldots, n-1))\, C^{2^1}(f_1; (0, 1, \ldots, n-1)) \cdots C^{2^{\lfloor \log(n) \rfloor}}(f_{\lfloor \log(n) \rfloor}; (0, 1, \ldots, n-1)). \quad (1)$$

For any $k = 0, 1, \ldots, \lfloor \log(n) \rfloor$, let $g := \mathrm{GCD}(n, 2^k)$ and $C_i := C(f_k; (i,\ i + 2^k \bmod n,\ \ldots, i + (\frac{n}{g} - 1)2^k \bmod n))$. Then by elementary modular arithmetic,

$$C^{2^k}(f_k; (0, 1, \ldots, n-1)) = C_0 C_1 \ldots C_{g-1},$$

and the targets of any two distinct C_i in this product do not overlap. This shows that each of the $\lfloor \log(n) \rfloor + 1$ factors in Eq. (1) can be written as a layer of C-type gates.

We next implement each of $\lfloor \log(n) \rfloor + 1$ layers of cyclic shift gates in Lemma 1 as circuits with $O(n)$ $C5$-type gates by expressing the cycles $(i_0, i_1, \ldots, i_{t-1})$ as products of length-5 cycles. Note that a length-5 cycle is always an even permutation and $(i_0, i_1, \ldots, i_{t-1})$ is an odd permutation when t is even. It is not possible to implement an odd permutation as a product of even permutations. However, with one exception, the C-type gates C_i come in pairs (recall that their number, g, is a power of two) and thus they can usually be paired up to form an even permutation that can then be decomposed

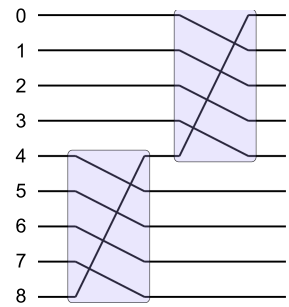

Fig. 2. Implementation of the 9-bit cyclic shift $C(f;(0,1,2,3,4,5,6,7,8))$ using the gates $C5(f;(4,5,6,7,8))$ and $C5(f;(0,1,2,3,4))$.

into a product of length-5 cycles. The one exception is the leftmost gate in Eq. (1), $C(f_0;(0,1,\ldots,n-1))$, when n is even. We handle this case first. □

Lemma 2. *$C(f_0;(0,1,\ldots,n-1))$ can be implemented by a reversible circuit with $O(n)$ elementary gates.*

Proof. The Boolean function $f_0(x)=x_1\oplus x_2\oplus\ldots\oplus x_n$ can be implemented on the top bit to control all bit SWAPs on the bottom bits, and it can be implemented on the bottom bit to control all bit SWAPs on the top bits. The number of controlled-SWAP gates required is $n-1$, and the total number of the CNOT gates required to compute/uncompute the control register is $4(n-1)$. We illustrated this construction in Fig. 3 for $n=7$. □

Lemma 3. *For $n\geq 5$:*

1) *for $t\leq 4$, pairs of two $C(f;(i_0,i_1,\ldots,i_{t-1}))$ gates can be implemented by an ancilla-free circuit using constantly many gates $C5(f;B)$;*
2) *for odd $t>4$ the $C(f;(i_0,i_1,\ldots,i_{t-1}))$ gate can be implemented by an ancilla-free circuit using $O(t)$ gates $C5(f;B)$.*
3) *for even $t>4$ pairs of $C(f;(i_0,i_1,\ldots,i_{t-1}))$ gates can be implemented by an ancilla-free circuit using $O(t)$ gates $C5(f;B)$;*

Proof. 1. There are three cases to consider: $t=2$, $t=3$, and $t=4$.

$t=2$. $C(f;(x_1,x_2))$ and $C(f;(y_1,y_2))$ can be implemented simultaneously by the circuit $C5(f;(y_1,x_1,y_2,a,x_2))$ $C5(f;(a,y_1,x_1,y_2,x_2))$. This is equivalent to saying that the following permutation equality holds: $(x_1,x_2)(y_1,y_2)=(y_1,x_1,y_2,a,x_2)(a,y_1,x_1,y_2,x_2)$.
Note that the bit 'a' can be found since $n\geq 5$. We will show only the permutation equalities in the rest of the proof, since it is trivial to translate those to circuits.

$t=3$. To implement a pair of gates $C(f;(x_1,x_2,x_3))$ and $C(f;(y_1,y_2,y_3))$ rely on the cycle product equality $(x_1,x_2,x_3)(y_1,y_2,y_3)=(x_1,y_1,x_2,y_2,y_3)(x_3,x_1,y_1,x_2,y_2)$.

$t=4$. Cycles (x_1,x_2,x_3,x_4) and (y_1,y_2,y_3,y_4) can be obtained by the equality

$$\begin{aligned}&(x_1,x_2,x_3,x_4)(y_1,y_2,y_3,y_4)\\&=(x_1,x_2)(x_1,x_3,x_4)\cdot(y_1,y_2)(y_1,y_3,y_4)\\&=(x_1,x_2)(y_1,y_2)\cdot(x_1,x_3,x_4)(y_1,y_3,y_4),\end{aligned}$$

where first and second part require two $C5$ gates each, as described in the cases $t=2$ and $t=3$, for a total of four $C5$ gates.**2.** The goal is to develop a circuit with $C5$ gates implementing the gate $C(f;(0,1,\ldots,t-1))$, where t is odd. There are two cases to consider, $t=4p+1$ and $t=4p+3$.

Case 1: $t=4p+1,\ p\geq 1$. We want to implement the integer permutation given by the cyclic shift $(0,1,\ldots,4p)$ by the cyclic shifts of length 5. This can be done as follows,

$$\begin{aligned}&(0,1,\ldots,4p)=(4p-4,4p-3,4p-2,4p-1,4p)\\&(4p-8,4p-7,4p-6,4p-5,4p-4)\cdots(0,1,2,3,4).\end{aligned}$$

This decomposition uses p length-5 cycles, resulting in the ability to implement $C(f;(0,1,\ldots,t-1))$ gate using $p=\frac{t-1}{4}$ $C5(f;B)$ gates. This construction is illustrated in Fig. 2 for $n=9$.

Case 2: $t=4p+3,\ p\geq 1$. Use the formula

$$\begin{aligned}&(0,1,\ldots,4p+2)=(4p,4p+1,4p+2)\cdot(0,1,\ldots,4p)\\&=(4p+2,4p,2,1,0)(4p+1,4p+2,0,1,2)\cdot(0,1,\ldots,4p).\end{aligned}$$

Since we already implemented $(0,1,\ldots,4p)$ with p $C5$ gates in Case 1 above, this implementation requires $\frac{t+5}{4}$ $C5$ gates.

3. The goal is to implement a pair of $C(f;(x_1,x_2,\ldots,x_t))$ and $C(f;(y_1,y_2,\ldots,y_t))$ where $t>4$ is even. Write

$$\begin{aligned}&(x_1,x_2,\ldots,x_t)\cdot(y_1,y_2,\ldots,y_t)\\&=(x_1,x_2)(x_1,x_3,x_4,\ldots,x_t)\cdot(y_1,y_2)(y_1,y_3,y_4,\ldots,y_t)\\&=(x_1,x_2)(y_1,y_2)\cdot(x_1,x_3,x_4,\ldots,x_t)\cdot(y_1,y_3,y_4,\ldots,y_t).\end{aligned}$$

Here, $(x_1,x_2)(y_1,y_2)$ requires two $C5$ gates per item **1.** case $t=2$, and each of $(x_1,x_3,x_4,\ldots,x_t)$ and $(y_1,y_3,y_4,\ldots,y_t)$ requires $O(t)$ gates per item **2.** □

Observe how the above proof implies that the number of $C5$ gates required to implement each of $C^{2^k}((0,1,\ldots,n-1);f_k)$ stages in Eq. (1) for $k=1,2,\ldots,\lfloor\log(n)\rfloor$ is between $\frac{n}{4}+Const$ and $\frac{n}{2}+Const$. Thus, per Lemma 2, the total number

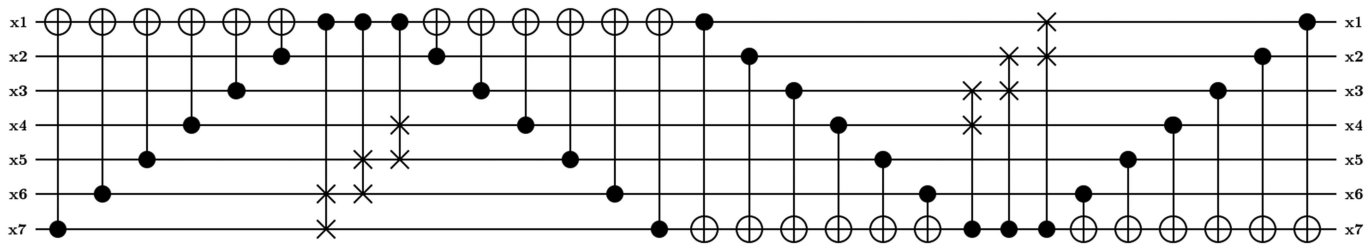

Fig. 3. Implementation of $C(f_0;(x_1,x_2,x_3,x_4,x_5,x_6,x_7))$, where $f_0(x)=x_1\oplus x_2\oplus x_3\oplus x_4\oplus x_5\oplus x_6\oplus x_7$.

of elementary and $C5$ gates required to implement **hwb** over n qubits is between $\frac{n\log(n)}{4}+O(n)$ and $\frac{n\log(n)}{2}+O(n)$.

We next show how to implement $C5(f_k;B)$ as a branching program, using Barrington's theorem [5], by closely following the original proof. In preparation for using Barringon's theorem, we first remove the dependence of the functions f_k in $C5(f_k;B)$ on the variables inside the set B, to allow the desired cyclic shift to be controlled by the values of $n-5$ variables outside the set B itself. To accomplish this, note that $C5(f_k;B)$ acts trivially on the strings 00000 and 11111; those can be ignored. This leaves 30 non-fixed by the operation 5-bit strings that can be partitioned into six disjoint subsets M_1, M_2, M_3, M_4, M_5, and M_6, with 5 strings each. Every subset M_i contains 5 cyclic shifts of some fixed 5-bit string, and is defined as follows:

$$\begin{aligned} M_1 &:= \{10000, 01000, 00100, 00010, 00001\},\\ M_2 &:= \{01111, 10111, 11011, 11101, 11110\},\\ M_3 &:= \{11000, 01100, 00110, 00011, 10001\},\\ M_4 &:= \{10100, 01010, 00101, 10010, 01001\},\\ M_5 &:= \{00111, 10011, 11001, 11100, 01110\},\\ M_6 &:= \{01011, 10101, 11010, 01101, 10110\}. \end{aligned} \tag{2}$$

We implement $C5(f_k;B)$ by performing the cyclic shifts of a single subset M_i per time.

First, let us introduce some more notations. Given a bit string $x\in\{0,1\}^n$, write $x=(y,b)$, where $b\in\{0,1\}^5$ is the restriction of x onto the register B and $y\in\{0,1\}^{n-5}$ is the rest of x. Let $w_i\in\{1,2,3,4\}$ be the Hamming weight of bit strings in M_i (note that all strings in the same subset M_i have the same weight). Define a Boolean function $f_{k,i}:\{0,1\}^{n-5}\to\{0,1\}$ such that $f_{k,i}(y)=1$ iff 2^k appears in the binary expansion of $|y|+w_i$. Then

$$f_k(x)=f_k(y,b)=f_{k,i}(y) \text{ for any } b\in M_i.$$

Define a gate $C5|_{M_i}(f_k;B):\{0,1\}^n\to\{0,1\}^n$ that maps an input $x=(y,b)$ to an output $x'=(y,b')$ according to the following rules:

- if $f_{k,i}(y)=0$ then $b'=b$;
- if $f_{k,i}(y)=1$ and $b\notin M_i$ then $b'=b$;
- if $f_{k,i}(y)=1$ and $b\in M_i$ then $b'\in M_i$ is obtained from b by cyclically shifting the elements of M_i.

By definition, the cyclic shift of bits in the register B can be realized by cyclically shifting elements of each subset M_i for $i=1,2,3,4,5,6$. Thus

$$C5(f_k(x);B)=\prod_{i=1}^{6} C5|_{M_i}(f_k(y);B). \tag{3}$$

Here the order in the product does not matter because the gates $C5|_{M_i}(f_k;B)$ pairwise commute. Note that the dependence of function f_k on the variables inside the set B has now been removed, and we can proceed to implementing $C5|_{M_i}(f_k;B)$ as a branching program, and finally mapping the instructions used by the branching program into reversible gates.

Recall some relevant notation used in Barrington's paper [5]. Let S_5 be the group of permutations of 5 numbers, $\{1,2,3,4,5\}$. Given a 5-tuple of distinct integers a_1,a_2,a_3,a_4,a_5, we write (a_1,a_2,a_3,a_4,a_5) to denote the 5-cycle. Let e be the identity permutation. A branching program of length L with m Boolean input variables $y_1,y_2,\ldots,y_m$ is a list of instructions $\langle y_i,\sigma_i,\tau_i\rangle$ with $i=1,2,\ldots,L$ and $\sigma_i,\tau_i\in S_5$, such that σ_i is applied if $y_i=1$, and τ_i is executed when $y_i=0$. Given a permutation $\sigma\in S_5$, the branching program is said to σ-compute a Boolean function $f(y)$ if executing the list of all instructions in the program results in e (the identity permutation) for all inputs y such that $f(y)=0$ and permutation σ for all inputs y such that $f(y)=1$.

Barrington's theorem asserts that any function in the class NC^1 can be $(1,2,3,4,5)$-computed by a branching program of polynomial size [5]. We next specialize the proof of the theorem to explicitly develop a short branching program that $(1,2,3,4,5)$-computes the Boolean function $f_{k,i}(y)$. Recall that $f_{k,i}(y)=1$ iff 2^k appears in the binary expansion of $y_1+y_2+\ldots+y_{n-5}+w_i$ with $w_i\in\{1,2,3,4\}$ being the weight of bit strings in M_i. It suffices to develop a branching program computing the Boolean function $f_k(y)$ with $y\in\{0,1\}^m$ and $m=n-5$ by appending two constant binary variables encoding w_i to the bit string y.

While the original proof [5] explored the mapping of logarithmic-depth classical circuits over $\{\mathsf{AND},\mathsf{OR}\}$ library, we focus on the classical circuits over 3-input 1-output $\mathsf{MAJ}(a,b,c):=ab\oplus bc\oplus ac$ and $\mathsf{XOR}(a,b,c):=a\oplus b\oplus c$ gates. Recall that the library $\{\mathsf{MAJ},\mathsf{XOR}\}$ is universal for classical computations if constant inputs are allowed.

Lemma 4. *Suppose y is an m-bit string and $f_k(y)$ is the kth bit in the binary representation of $W=y_1+y_2+\ldots+y_m$. The function $f_k(y)$ can be $(1,2,3,4,5)$-computed by a branching program of size $O(m^{5.42})$.*

Proof. First, we describe a logarithmic-depth classical circuit that computes functions $f_k(y)$ for the range of applicable values k, and second, report expressions for MAJ and XOR in the form of a branching program that can be used in the recursion [5, Proof of Theorem 1]. The length of the branching program computing $f_k(y)$ is upper bounded by taking the maximal length of the program implementing MAJ or XOR to the power of the circuit depth.

First, construct a classical circuit with MAJ and XOR gates that implements $f_k(y)$. To do so, we develop a circuit that computes all bits of the $W(y)$, and for the purpose of implementing a given single Boolean component, discard all gates that compute the bits we are not interested in. Such operation does not increase the depth of the circuit, and may, in fact, decrease it slightly.

To find $W(y)$, we employ a circuit consisting of two stages. First, compose a circuit of depth $\log_{3/2}(m)+O(1)$ with 3-input 2-output Full Adder gates $\mathsf{FA}(a,b,c):=(\mathsf{MAJ}(a,b,c),\mathsf{XOR}(a,b,c))$ by grouping as many triples of digits of same significance at each step as possible (note that MAJ and XOR are implemented in parallel). We finish this first stage when the output contains two $\log(m)$-digit integer numbers u and v such that $W=u+v$. To analyze this circuit, it is convenient to group all bits needing to be added into the smallest set of integer numbers, and count the reduction in the number of integers left to be added by treating layers of FA gates as Carry-Save Adders [21], [22]. A Carry-Save Adder is

defined as the 3-integer into 2-integer adder, which is implemented by applying the Full Adders to the individual components of the three integer numbers at the input. Since the number of integers left to be added changes by a factor of $\frac{2}{3}$ at each step, and every step is implemented by a depth-1 MAJ/XOR circuit, the depth of the first stage is $\log_{3/2}(m)+O(1)$. To find the individual components of $W(y)$, the second stage adds two $\log(m)$-digit integer numbers u and v. This can be accomplished by any logarithmic-depth integer addition circuit in depth $O(\log\log(m))$, such as [23]. The total depth is thus $\log_{3/2}(m)+O(\log\log(m))$.

Next, construct S_5-programs computing the MAJ and XOR functions:

$$\begin{aligned}&\langle z_1,(1,4,3,2,5),e\rangle\ \langle z_2,(1,3,5,4,2),e\rangle\ \langle z_3,(1,2,5,3,4),e\rangle\\ &\langle z_1,(1,2,3,4,5),e\rangle\ \langle z_2,(1,2,4,5,3),e\rangle\ \langle z_3,(1,4,3,5,2),e\rangle\\ &\qquad\langle z_1,(1,5,4,3,2),e\rangle\ \langle z_1,(1,5,2,3,4),e\rangle\\ &=\begin{cases} e & \text{if}\quad \mathsf{MAJ}(z_1,z_2,z_3)=0\\ (1,2,3,4,5) & \text{if}\quad \mathsf{MAJ}(z_1,z_2,z_3)=1,\end{cases}\end{aligned}\tag{4}$$

$$\begin{aligned}&\langle z_2,(1,2,3,5,4),e\rangle\ \langle z_3,(1,2,4,5,3),e\rangle\ \langle z_2,(1,3,5,4,2),e\rangle\\ &\langle z_3,(1,4,5,3,2),e\rangle\ \langle z_1,(1,2,3,4,5),e\rangle\ \langle z_2,(1,3,4,2,5),e\rangle\\ &\langle z_2,(1,3,2,4,5),e\rangle\ \langle z_3,(1,3,4,2,5),e\rangle\ \langle z_3,(1,3,2,4,5),e\rangle\\ &=\begin{cases} e & \text{if}\quad \mathsf{XOR}(z_1,z_2,z_3)=0\\ (1,2,3,4,5) & \text{if}\quad \mathsf{XOR}(z_1,z_2,z_3)=1.\end{cases}\end{aligned}\tag{5}$$

The branching program that $(1,2,3,4,5)$-computes $f_k(y)$ is created by recursively replacing gates MAJ and XOR in the circuit constructed above with the branching programs Eqs. (4) and (5), where each z_i is either one of the primary input variables $y_1,y_2\ldots,y_m$ or one of the intermediate variables in the circuit computing $f_k(y)$, until all instructions are controlled by constants and primary variables $y_1,y_2\ldots,y_m$. The recoding of branches of the program τ-computing a desired intermediate variable z_* when $\tau\neq(1,2,3,4,5)$ (note how Eqs. (4) and (5) $(1,2,3,4,5)$-compute the gates, but not τ-compute them for arbitrary τ) is accomplished in accordance with [5, Lemma 1]. The total length of the branching program is thus upper bounded by the size of longest branching program implementation of the basic gates used (MAJ and XOR) raised to the power the depth of the circuit it encodes,

$$\begin{aligned}9^{\log_{3/2}(m)+O(\log\log(m))}&=O(m^{5.4190225\ldots}\log(m)^{O(1)})\\&=O(m^{5.42}).\end{aligned}$$

□

We conclude this section by summarizing the main result in a Theorem.

Theorem 1. *The n-bit* **hwb** *function can be implemented by an ancilla-free reversible circuit of size $O(n^{6.42})$.*

Proof. First, implement each instruction $\langle z_*,(a_1,a_2,a_3,a_4,a_5),e\rangle$ where z_* is either a primary variable or a constant and the sets $\{a_1,a_2,a_3,a_4,a_5\}$ are defined per Eq. (2), using constantly many basic reversible gates. This can be accomplished by employing a reversible logic synthesis algorithm, e.g., [10]. Next, use Lemma 4 with $m=n-5$ and $x=y\sqcup B$ to implement all necessary $C5|_{M_i}(f_k(y);B)$ gates, using a branching program with

$$O\Big(9^{\log_{3/2}(n-5)+O(\log\log(n-5))}\Big)=O\Big(9^{\log_{3/2}(n)+O(\log\log(n))}\Big),$$

instructions. Each such branching program requires $O(9^{\log_{3/2}(n)+O(\log\log(n))})$ basic reversible gates since every instruction requires constantly many basic reversible gates. Use six $C5|_{M_i}(f_k(y);B)$ gates to implement one $C5(f_k(x);B)$ gate, using Eq. (3). Each $C5(f_k(x);B)$ thus costs $O(9^{\log_{3/2}(n)+O(\log\log(n))})$ basic reversible gates. Combine Lemmas 1, 2, and 3 to implement **hwb** using $O(n\log(n))$ $C5(f_k(x);B)$ gates, implying the total basic reversible gate count of

$$\begin{aligned}&O\Big(9^{\log_{3/2}(n)+O(\log\log(n))}\cdot n\log(n)\Big)\\&=O\Big(n^{6.4190225\ldots}\log(n)^{O(1)}\Big)=O(n^{6.42}).\end{aligned}$$

□

4 Ancilla-Free Quantum Circuit of Size $O(n^2)$

Consider a register of n qubits and let C be the cyclic shift operator,

$$\mathsf{C}|x_1,x_2,x_3,\ldots,x_n\rangle=|x_n,x_1,x_2,\ldots,x_{n-1}\rangle.$$

The hidden weighted bit function $U_{\mathbf{hwb}}$ may be written as

$$U_{\mathbf{hwb}}|x\rangle=\mathsf{C}^{x_1+x_2+\ldots+x_n}|x\rangle\quad\text{for all } x\in\{0,1\}^n.$$

In other words, $U_{\mathbf{hwb}}$ implements the kth power of C on the subspace with the Hamming weight k. Here we show that $U_{\mathbf{hwb}}$ can be implemented by an ancilla-free quantum circuit of the size $O(n^2)$. The circuit is expressed using Clifford gates and single-qubit Z-rotations.

Let $W:=\sum_{j=0}^{n-1}|1\rangle\langle 1|_j$ be the Hamming weight operator. Our starting point is

Lemma 5. *Suppose $\mathsf{C}=e^{iH}$ for some n-qubit Hamiltonian H that commutes with W. Then*

$$U_{\mathbf{hwb}}=e^{iHW}.$$

Proof. Indeed, let $\mathcal{L}_k$ be the subspace spanned by all basis states $|x\rangle$ with the Hamming weight k. The full Hilbert space of n qubits is the direct sum $\mathcal{L}_0\oplus\mathcal{L}_1\oplus\ldots\oplus\mathcal{L}_n$. Let us say that an operator O is block-diagonal if O maps each subspace $\mathcal{L}_k$ into itself. Since H commutes with W, we infer that H is block-diagonal. Therefore HW and e^{iHW} are also block-diagonal. Note that HW and kH have the same restriction onto $\mathcal{L}_k$. Thus e^{iHW} and e^{ikH} have the same restriction onto $\mathcal{L}_k$. By assumption, $e^{iH}=\mathsf{C}$. Thus e^{iHW} and C^k have the same restriction onto $\mathcal{L}_k$. Likewise, $U_{\mathbf{hwb}}$ is block-diagonal and the restriction of $U_{\mathbf{hwb}}$ onto $\mathcal{L}_k$ is C^k. We conclude that $U_{\mathbf{hwb}}$ and e^{iHW} have the same restriction onto $\mathcal{L}_k$ for all k. Since both operators are block-diagonal, one has $U_{\mathbf{hwb}}=e^{iHW}$. □

To construct a Hamiltonian H satisfying conditions of Lemma 5 it is natural to work in the momentum space [17], [18] such that the cyclic shift operator C becomes diagonal. Unfortunately, the momentum space is not well-defined for qubits and Pauli operators. Instead, we will use the the language of fermions and the fermionic Fourier transform [19], [20]. First, define fermionic creation and annihilation operators $a_p^\dagger$ and a_p with $p \in \mathbb{Z}_n := \{0, 1, \ldots, n-1\}$ as

$$a_p^\dagger = \underbrace{Z \otimes Z \otimes \cdots \otimes Z}_{p} \otimes |1\rangle\langle 0| \otimes \underbrace{I \otimes I \otimes \cdots \otimes I}_{n-p-1}$$

$$a_p = \underbrace{Z \otimes Z \otimes \cdots \otimes Z}_{p} \otimes |0\rangle\langle 1| \otimes \underbrace{I \otimes I \otimes \cdots \otimes I}_{n-p-1}.$$

Here $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ is the Pauli-Z operator. The creation and annihilation operators obey the fermionic canonical commutation rules, $a_p a_q = -a_q a_p$ and $a_p a_q^\dagger + a_q^\dagger a_p = \delta_{p,q} I$. We will make use of the ability to perform unitary basis changes in the space of fermionic operators. Namely, suppose u is a unitary $n \times n$ matrix. Then there exists an n-qubit unitary operator U such that $U a_p U^\dagger = \sum_{q \in \mathbb{Z}_n} u_{p,q} a_q$ for all $p \in \mathbb{Z}_n$. Furthermore, U can be implemented by an ancilla-free quantum circuit of size $O(n^2)$, see [20]. The desired momentum space basis is defined by choosing U as the Fermionic Fourier Transform [20].

Definition 1. *A Fermionic Fourier Transform is a unitary n-qubit operator F such that $\mathsf{F}|0^n\rangle = |0^n\rangle$ and*

$$\mathsf{F} a_p \mathsf{F}^\dagger = \frac{1}{\sqrt{n}} \sum_{q \in \mathbb{Z}_n} e^{2\pi i p q / n} a_q \qquad \text{for all } p \in \mathbb{Z}_n. \tag{6}$$

Note that Eq. (6) uniquely specifies F. Indeed, suppose $x \in \{0,1\}^n$ is a weight-k basis state with ones at qubits $p_1 < p_2 < \ldots < p_k$. Then

$$\mathsf{F}\,|x\rangle = \mathsf{F} a_{p_1}^\dagger a_{p_2}^\dagger \cdots a_{p_k}^\dagger |0^n\rangle = \mathsf{F} a_{p_1}^\dagger a_{p_2}^\dagger \cdots a_{p_k}^\dagger \mathsf{F}^\dagger |0^n\rangle = \prod_{i=1}^{k} \mathsf{F} a_{p_i}^\dagger \mathsf{F}^\dagger |0^n\rangle. \tag{7}$$

Since each operator $\mathsf{F} a_{p_i}^\dagger \mathsf{F}^\dagger = (\mathsf{F} a_{p_i} \mathsf{F}^\dagger)^\dagger$ is determined by Eq. (6), this uniquely specifies the action of F on the basis vectors $|x\rangle$. It will be important that F commutes with the Hamming weight operator W,

$$\mathsf{F} W = W \mathsf{F}. \tag{8}$$

Indeed, from Eqs. (6) and (7) one can see that $\mathsf{F}|x\rangle$ is a linear combination of states $a_{q_1}^\dagger a_{q_2}^\dagger \cdots a_{q_k}^\dagger |0^n\rangle$. Since $(a_q^\dagger)^2 = 0$, the state $a_{q_1}^\dagger a_{q_2}^\dagger \cdots a_{q_k}^\dagger |0^n\rangle$ is non-zero only if all indices $q_1, q_2, \ldots, q_k$ are distinct. Such state has weight k. Thus F maps weight-k states to linear combinations of weight-k states proving Eq. (8).

We will use the following fact established by Kivlichan *et al.* [20].

Lemma 6. *The fermionic Fourier transform F on n qubits can be implemented by a quantum circuit of size $O(n^2)$. The circuit requires no ancillary qubits.*

For completeness, we provide a simplified proof of Lemma 6 and an explicit construction of the quantum circuit realizing F in Section 4.1. Now we are ready to define a Hamiltonian H satisfying conditions of Lemma 5. Let

$$E = \frac{1}{2}(I + Z^{\otimes n}),$$

be the projector onto the even-weight subspace. Define n-qubit Hamiltonians

$$H_0 := \frac{2\pi}{n} \sum_{p \in \mathbb{Z}_n} p |1\rangle\langle 1|_p, \quad H' := H_0 + \frac{\pi}{n} WE, \text{ and}$$

$$H := V^\dagger H' V, \text{ where } V = \mathsf{F}^\dagger e^{i H_0 E/2}. \tag{9}$$

Lemma 7. *The Hamiltonian H defined in Eq. (9) satisfies $C = e^{iH}$.*

A proof of this lemma is given in Section 4.2. A high-level intuition behind the definition of H comes from the fact that $\mathsf{F} H_0 \mathsf{F}^\dagger$ is the fermionic momentum operator. Note that $H = \mathsf{F} H_0 \mathsf{F}^\dagger$ in the odd-weight subspace where $E = 0$. The extra terms in the definition of H are needed to change integer momentums (periodic boundary conditions) in the odd-weight subspace to half-integer momentums (anti-periodic boundary conditions) in the even-weight subspace. This accounts for the difference between the qubit cyclic shift and its fermionic analogue, as detailed in Section 4.2.

From Eq. (8) one can see that $HW = WH$. Thus H satisfies conditions of Lemma 5. Combining Lemma 5, Lemma 7, and noting that $VW = WV$ one arrives at

$$\begin{aligned} U_{\mathbf{hwb}} &= e^{iHW} = e^{iV^\dagger H' V W} = V^\dagger e^{iH'W} V \\ &= e^{-iH_0 E/2} \mathsf{F} e^{iH'W} \mathsf{F}^\dagger e^{iH_0 E/2}. \end{aligned} \tag{10}$$

Here we used the well-known fact that $e^{iV^\dagger O V} = V^\dagger e^{iO} V$ for any Hermitian operator O and any unitary V (which can be verified by expanding the exponent using the Taylor series and noting that $(V^\dagger O V)^p = V^\dagger O^p V$ for all $p \geq 1$). We claim that each term in Eq. (10) can be implemented using $O(n^2)$ two-qubit gates without ancillary qubits. By Lemma 6, the layers F and $\mathsf{F}^\dagger$ have gate cost $O(n^2)$.

For the term $e^{iH_0 E/2}$ and its inverse, we have the following lemma.

Lemma 8. *The operator $e^{iH_0 E/2}$ can be implemented by a quantum circuit of size $O(n)$ without using ancillary qubits.*

Proof. If we set $\theta_p = p\pi/n$, then

$$e^{iH_0 E/2} = R_1 R_2 \cdots R_{n-1}, \quad \text{where } R_p = e^{i\theta_p |1\rangle\langle 1|_p E}. \tag{11}$$

The operator $|1\rangle\langle 1|_p E$ projects the subset of qubits $\mathbb{Z}_n \backslash \{p\}$ onto the odd-weight subspace. Note $p \neq 0$ and let C_p be a CNOT circuit that computes the parity of $\mathbb{Z}_n \backslash \{p\}$ into the

qubit 0,

$$C_p = \prod_{j\in\mathbb{Z}_n\setminus\{0,p\}} \mathrm{CNOT}_{j,0}.$$

Then $|1\rangle\langle 1|_p E = C_p^\dagger |11\rangle\langle 11|_{0p} C_p$ and thus

$$R_p = C_p^\dagger e^{i\theta_p |11\rangle\langle 11|_{0p}} C_p.$$

Therefore, an individual R_p is implemented with $O(n)$ gates, which suggests $e^{iH_0E/2}$ can be implemented with $O(n^2)$ gates. However, we can improve this count by noting that for $p\neq q$ $C_pC_q^\dagger = \mathrm{CNOT}_{p,0}\mathrm{CNOT}_{q,0}$. Thus, in fact, the product in Eq. (11) can be implemented with just $O(n)$ gates. □

We still need to implement the term $e^{iH'W} = e^{iH_0W} e^{i(\pi/n)W^2E}$. The operator e^{iH_0W} is a product of $O(n^2)$ rotations $e^{i\theta|11\rangle\langle 11|}$ and $e^{i\theta|1\rangle\langle 1|}$. Although a naive implementation of $e^{i(\pi/n)W^2E}$ requires $O(n^3)$ gates, we next show that a better implementation exists.

Lemma 9. *The operator $e^{i(\pi/n)W^2E}$ can be implemented by a quantum circuit of size $O(n^2)$ without using ancillary qubits.*

Proof. First, note that

$$\begin{aligned} W^2E = 2 \sum_{p,p'\in\mathbb{Z}_n 0<p<p'} |11\rangle\langle 11|_{pp'}E \\ + 2\sum_{p\in\mathbb{Z}_n 0<p} |11\rangle\langle 11|_{0p}E + \sum_{p\in\mathbb{Z}_n} |1\rangle\langle 1|_p E. \end{aligned} \tag{12}$$

The terms in Eq. (12) commute. Therefore, we have, with arbitrary order within the products,

$$e^{i(\pi/n)W^2E} = \prod_{p,p'\in\mathbb{Z}_n 0<p<p'} U_{pp'} \prod_{p\in\mathbb{Z}_n 0<p} U_{0p} \prod_{p\in\mathbb{Z}_n} U_p, \tag{13}$$

where, for $p < p'$,

$$U_{pp'} = e^{i(2\pi/n)|11\rangle\langle 11|_{pp'}E} \text{ and } U_p = e^{i(\pi/n)|1\rangle\langle 1|_p E}.$$

The second and third products in Eq. (13) can be implemented with $O(n)$ gates using arguments similar to those in Lemma 8. In the rest of this proof we focus on the first product and show that it can be implemented with $O(n^2)$ gates.

Notice that $|11\rangle\langle 11|_{pp'}E$ projects the subset of qubits $\mathbb{Z}_n\setminus\{p,p'\}$ onto the even weight subspace while projecting qubits p and p' to $|11\rangle_{pp'}$. Therefore, if $0 < p < p'$, we can define $S_{pp'} := \mathbb{Z}_n\setminus\{0,p,p'\}$ and

$$C_{pp'} := \prod_{j\in S_{pp'}} \mathrm{CNOT}_{j,0},$$

such that

$$U_{pp'} = C_{pp'}^\dagger e^{i(2\pi/n)|011\rangle\langle 011|_{0pp'}} C_{pp'}.$$

This implementation of $U_{pp'}$ takes $O(n)$ gates, which suggests $O(n^3)$ gates might be needed to implement all $n(n-1)/2$ factors in the first product in Eq. (13). However, we can order the factors in such a way as to allow massive cancellation between consecutive CNOT circuits $C_{pp'}$ and implement the first product with just $O(n^2)$ total gates.

Notice that

$$C_{pp'}C_{qq'}^\dagger = \prod_{j\in S_{pp'}\Delta S_{qq'}} \mathrm{CNOT}_{j,0},$$

is a circuit of at most four CNOT gates. In fact, it is a circuit with just two CNOT gates when $|\{p,p'\}\cap\{q,q'\}| = 1$. Thus, the following two products can be implemented with $O(n)$ gates:

$$U_{x\uparrow} = U_{x,x+1}U_{x,x+2}\cdots U_{x,n-1}, U_{y\downarrow} = U_{y,n-1}U_{y,n-2}\cdots U_{y,y+1},$$

where $x, y \in \mathbb{Z}_n\setminus\{n-1\}$. Hence, the first product in Eq. (13) can be implemented with $O(n^2)$ gates because

$$\prod_{p,p'\in\mathbb{Z}_n 0<p<p'} U_{pp'} = U_{1\uparrow}U_{2\downarrow}U_{3\uparrow}\cdots U_{n-2,n-1}.$$

□

The above implementation of $e^{i(\pi/n)W^2E}$ requires three-qubit gates of the form $e^{i\theta|011\rangle\langle 011|}$. The latter can be decomposed into a sequence of $O(1)$ two-qubit Clifford gates and single-qubit Z-rotations using the standard methods [24]. We summarize main result of this section in the following Theorem.

Theorem 2. *Eq. (10) reports an ancilla-free quantum circuit of size $O(n^2)$ implementing $U_{\mathbf{hwb}}$.*

The next two subsections detail various proofs building up to Theorem 2; an uninterested reader may skip to Section 5.

4.1 Implementation of the fermionic Fourier transform

Here we use the method of Ref. [20] to construct a quantum circuit implementing the fermionic Fourier transform F on n qubits and illustrate it for $n = 3$. The circuit is expressed using $O(n^2)$ single-qubit and two-qubit gates

$$\mathsf{S}(\gamma) := e^{i\gamma|1\rangle\langle 1|} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\gamma} \end{bmatrix},$$

and

$$\begin{aligned} \mathsf{R}(\alpha,\beta) &:= e^{\alpha e^{i\beta}|10\rangle\langle 01| - \alpha e^{-i\beta}|01\rangle\langle 10|} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -e^{-i\beta}\sin(\alpha) & 0 \\ 0 & e^{i\beta}\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned}$$

Here α, β, and γ are real parameters. We use subscripts $p, q \in \mathbb{Z}_n$ to indicate qubits acted upon by each gate. In the fermionic language, $\mathsf{R}_{p,p+1}(\alpha,\beta)$ implements a Givens rotation in the two-dimensional subspace spanned by operators a_p and a_{p+1}. Namely, let $\mathsf{R}_{p,p+1} = \mathsf{R}_{p,p+1}(\alpha,\beta)$. Then

$$\mathsf{R}_{p,p+1}a_p\mathsf{R}_{p,p+1}^\dagger = \cos(\alpha)a_p - \sin(\alpha)e^{i\beta}a_{p+1}, \tag{14}$$

$$\mathsf{R}_{p,p+1}a_{p+1}\mathsf{R}_{p,p+1}^\dagger = \sin(\alpha)e^{-i\beta}a_p + \cos(\alpha)a_{p+1}. \tag{15}$$

We also need a fermionic SWAP gate [20], [25] defined as

$$\mathsf{fSWAP} = \mathsf{CZ} \cdot \mathsf{SWAP} = \mathsf{R}(\pi/2, \pi/2)\mathsf{S}(-\pi/2)^{\otimes 2}.$$

One can easily check that $(\mathsf{fSWAP}_{p,p+1})a_p(\mathsf{fSWAP}_{p,p+1})^\dagger = a_{p+1}$ and $(\mathsf{fSWAP}_{p,p+1})a_{p+1}(\mathsf{fSWAP}_{p,p+1})^\dagger = a_p$. Define a unitary $n \times n$ matrix f with matrix elements

$$f_{p,q} = n^{-1/2} e^{2\pi i p q/n}, \text{ where } p, q \in \mathbb{Z}_n. \tag{16}$$

We will write $\mathsf{row}(f, p)$ for the pth row of f. Below we define a function $\mathsf{ColumnReduce}(f, m, U)$ that takes as input a unitary $n \times n$ matrix f, an integer $m \in \mathbb{Z}_n$, and a quantum circuit U acting on n qubits. The function returns a modified unitary matrix f' and a modified quantum circuit U'. A quantum circuit realizing the fermionic Fourier transform F on n qubits is generated by the following algorithm.

Algorithm 1. FermionicFourierTransform

1: Let f be the $n \times n$ unitary matrix defined in Eq. (16)
2: $U \leftarrow I$ ▷ Empty quantum circuit
3: **for** $m = n - 1$ to 0 **do**
4: $(f, U) = \mathsf{ColumnReduce}(f, m, U)$
5: **end for**
6: **return** $\mathsf{F} = U^{-1}$

Algorithm 2. $\mathsf{ColumnReduce}(f, m, U)$

1: **for** $p = 0$ to $m - 1$ **do**
2: **if** $f_{p,m} \neq 0$ **or** $f_{p+1,m} \neq 0$ **then**
3: **if** $f_{p+1,m} = 0$ **then**
4: Swap $\mathsf{row}(f, p)$ and $\mathsf{row}(f, p+1)$
5: $U \leftarrow \mathsf{fSWAP}_{p,p+1} \cdot U$ ▷ Add fSWAP gate
6: **end if** ▷ Now $f_{p+1,m} \neq 0$
7: Choose angles α, β such that $\tan(\alpha)e^{-i\beta} = -f_{p,m}/f_{p+1,m}$
8: $v \leftarrow \mathsf{row}(f, p)$
9: $\mathsf{row}(f, p) \leftarrow \cos(\alpha)\mathsf{row}(f, p) + \sin(\alpha)e^{-i\beta}\mathsf{row}(f, p+1)$ ▷ Now $f_{p,m} = 0$
10: $\mathsf{row}(f, p+1) \leftarrow \cos(\alpha)\mathsf{row}(f, p+1) - \sin(\alpha)e^{i\beta}v$
11: $U \leftarrow \mathsf{R}_{p,p+1}(\alpha, \beta) \cdot U$ ▷ Add R gate
12: **end if**
13: **end for** ▷ Now $f_{m,m}$ is the only nonzero in the mth column of f
14: $\gamma \leftarrow \text{phase}(f_{m,m})$ ▷ Now $f_{m,m} = e^{i\gamma}$
15: $f_{m,m} = 1$
16: $U \leftarrow \mathsf{S}_m(\gamma) \cdot U$ ▷ Add S gate
17: **return** (f, U)

We claim that the quantum circuit U and the unitary matrix f obtained after each call to the function *ColumnReduce* have the property

$$(U\mathsf{F})a_p(U\mathsf{F})^\dagger = \sum_{q \in \mathbb{Z}_n} f_{p,q} a_q \text{ for all } p \in \mathbb{Z}_n. \tag{17}$$

Indeed, Eq. (17) is trivially true initially when $U = I$ and f is defined by Eq. (16). The lines 4 and 7-10 of Algorithm 2 apply a sequence of Givens rotations to the matrix f setting to zero all matrix elements $f_{p,m}$ with $0 \leq p < m$ and setting $f_{m,m} = 1$. The order in which matrix elements of f are set to 0 or 1 is illustrated for $n = 3$ below (asterisks indicate matrix elements of f):

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \to \begin{bmatrix} * & * & 0 \\ * & * & * \\ * & * & * \end{bmatrix} \to \begin{bmatrix} * & * & 0 \\ * & * & 0 \\ * & * & * \end{bmatrix} \to \begin{bmatrix} * & * & 0 \\ * & * & 0 \\ * & * & 1 \end{bmatrix} \to \begin{bmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & 1 \end{bmatrix} \to \begin{bmatrix} * & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix}. \tag{18}$$

Since f remains unitary at each step, the final unit-diagonal low-triangular matrix is the identity, i.e., $f = I$ after the last iteration of Algorithm 1. Each time a Givens rotation is applied to some rows $p, p+1$ of the matrix f, the corresponding Givens rotations of fermionic operators a_p, a_{p+1} are added to the quantum circuit U, see Eqs. (14) and (15). More precisely, the angles α, β at Line 7 are chosen such that the operator

$$\mathsf{R}_{p,p+1}(\alpha, \beta)(f_{p,m}a_p + f_{p+1,m}a_{p+1})\mathsf{R}_{p,p+1}(\alpha, \beta)^\dagger,$$

is proportional to a_{p+1}, see Eqs. (14) and 15). Thus the property Eq. (17) is maintained at each step. After the last iteration of Algorithm 1 one has $f = I$ and Eq. (17) gives $(U\mathsf{F})a_p(U\mathsf{F})^\dagger = a_p$ for all p. Furthermore $U|0^n\rangle = |0^n\rangle$ since all gates added to U map $|0^n\rangle$ to itself. We conclude that $U = \mathsf{F}^{-1}$ after the last iteration of Algorithm 1. Thus the algorithm returns a quantum circuit realizing F. The inverse circuit U^{-1} can be obtained from U using the identities $\mathsf{R}(\alpha, \beta)^{-1} = \mathsf{R}(-\alpha, \beta)$ and $\mathsf{S}(\gamma)^{-1} = \mathsf{S}(-\gamma)$. The direct inspection shows that the total numberof gates fSWAP, R and S added to U is $O(n^2)$. We implemented Algorithms 1 and 2 in Matlab obtaining the circuit illustrated in Fig. 4 in the case $n = 3$.

4.2 Proof of Lemma 7

First note that

$$\mathsf{C} = \mathsf{SWAP}_{0,1}\mathsf{SWAP}_{1,2} \cdots \mathsf{SWAP}_{n-2,n-1}.$$

Define a fermionic cyclic shift

$$\mathsf{fC} = \mathsf{fSWAP}_{0,1}\mathsf{fSWAP}_{1,2} \cdots \mathsf{fSWAP}_{n-2,n-1}. \tag{19}$$

A simple algebra shows that

$$\mathsf{C}|x\rangle = (-1)^{x_{n-1}(x_0 + x_1 + \ldots + x_{n-2})}\mathsf{fC}|x\rangle.$$

Let $k = x_0 + x_1 + \ldots + x_{n-1}$ be the Hamming weight of x. Then

$$\begin{aligned}(-1)^{x_{n-1}(x_0+x_1+\ldots+x_{n-2})} &= (-1)^{x_{n-1}(k-x_{n-1})} \\ &= (-1)^{x_{n-1}k + x_{n-1}} = (-1)^{x_{n-1}(k+1)}.\end{aligned}$$

Thus $\mathsf{C} = \mathsf{fC}$ on the odd-weight subspace and $\mathsf{C} = \mathsf{fC}Z_{n-1}$ on the even-weight subspace, i.e.,

$$\mathsf{C} = E\mathsf{fC}Z_{n-1} + (I - E)\mathsf{fC}. \tag{20}$$

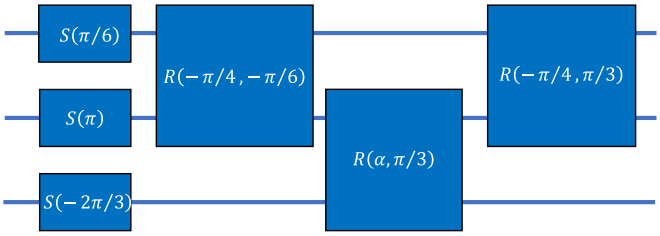

Fig. 4. Quantum circuit realizing the 3-qubit fermionic Fourier transform F. The circuit was generated using Algorithm 1. Here $\alpha = -(1/2)\arccos(1/3) \approx -0.9553$.

We claim that

$$\mathsf{fC} = \mathsf{F}e^{iH_0}\mathsf{F}^\dagger. \tag{21}$$

Indeed, let $G := \mathsf{F}e^{iH_0}\mathsf{F}^\dagger$. First note that $\mathsf{fC}|0^n\rangle = G|0^n\rangle = |0^n\rangle$. Since any state can be obtained from $|0^n\rangle$ by applying the creation operators $a_p^\dagger$, it suffices to check that

$$\mathsf{fC}^\dagger a_p \mathsf{fC} = G^\dagger a_p G,$$

for all p. Recall that

$$(\mathsf{fSWAP}_{p,p+1})a_p(\mathsf{fSWAP}_{p,p+1})^\dagger = a_{p+1} \text{ and}$$
$$(\mathsf{fSWAP}_{p,p+1})a_{p+1}(\mathsf{fSWAP}_{p,p+1})^\dagger = a_p.$$

Combining this and Eq. (19) one gets $\mathsf{fC}^\dagger a_p \mathsf{fC} = a_{p-1}$, where the indices of fermionic operators are evaluated modulo n. Using the identities

$$e^{-iH_0}a_q e^{iH_0} = e^{2\pi iq/n}a_q,$$

$$\mathsf{F}a_p\mathsf{F}^\dagger = \frac{1}{\sqrt{n}}\sum_{q\in\mathbb{Z}_n}^{n} e^{2\pi ipq/n}a_q, \text{ and}$$

$$\mathsf{F}^\dagger a_q \mathsf{F} = \frac{1}{\sqrt{n}}\sum_{r\in\mathbb{Z}_n}^{n} e^{-2\pi iqr/n}a_r,$$

one gets

$$G^\dagger a_p G = n^{-1/2}\sum_{q\in\mathbb{Z}_n} e^{2\pi i(1-p)q/n}\mathsf{F}a_q\mathsf{F}^\dagger$$
$$= n^{-1}\sum_{q,r\in\mathbb{Z}_n} e^{2\pi i(1-p+r)q/n}a_r = a_{p-1}.$$

Thus $G^\dagger a_p G = \mathsf{fC}^\dagger a_p \mathsf{fC} = a_{p-1}$, proving Eq. (21).

Next we claim that

$$\mathsf{fC}Z_{n-1} = e^{-iH_0/2}\mathsf{fC}e^{iH_0/2}e^{i(\pi/n)W}. \tag{22}$$

Indeed, let $L := \mathsf{fC}Z_{n-1}$ and $R := e^{-iH_0/2}\mathsf{fC}e^{iH_0/2}e^{i(\pi/n)W}$. Since $L|0^n\rangle = R|0^n\rangle = |0^n\rangle$, it suffices to check that $L^\dagger a_p L = R^\dagger a_p R$ for all $p \in \mathbb{Z}_n$. A simple algebra gives

$$e^{-i(\pi/n)W}a_p e^{i(\pi/n)W} = e^{i(\pi/n)}a_p,$$

$$Z_{n-1}a_pZ_{n-1} = \begin{cases} a_p & \text{if} \quad 0 \le p \le n-2 \\ -a_p & \text{if} \quad p = n-1 \end{cases},$$

$$e^{iH_0/2}a_p e^{-iH_0/2} = e^{-i\pi p/n}a_p, \text{ and}$$

$$e^{-iH_0/2}a_p e^{iH_0/2} = e^{i\pi p/n}a_p,$$

for all $p \in \mathbb{Z}_n$. Recall that $\mathsf{fC}^\dagger a_p \mathsf{fC} = a_{p-1}$. Using the above identities one gets

$$L^\dagger a_p L = \begin{cases} a_p & \text{if} \quad 1 \le p \le n-1 \\ -a_p & \text{if} \quad p = 0 \end{cases}$$

and

$$R^\dagger a_p R = e^{-i\pi p/n}e^{i\pi p'/n}e^{i(\pi/n)}a_{p-1},$$

where $p' \equiv p-1 \pmod n$. Note that

$$e^{-i\pi p/n}e^{i\pi p'/n} = \begin{cases} e^{-i\pi/n} & \text{if} \quad 1 \le p \le n-1 \\ -e^{-i\pi/n} & \text{if} \quad p = 0 \end{cases}.$$

Thus $L^\dagger a_p L = R^\dagger a_p R$, that is, $L = R$, proving Eq. (22).

Combining Eqs. (20),(21), and (22) one infers that the restrictions of C onto the odd-weight and even-weight subspaces coincide with the operators $\mathsf{C}_{odd} = \mathsf{F}e^{iH_0}\mathsf{F}^\dagger$ and $\mathsf{C}_{even} = e^{-iH_0/2}(\mathsf{F}e^{iH_0}\mathsf{F}^\dagger)e^{iH_0/2}e^{i(\pi/n)W}$ respectively. Thus

$$\mathsf{C} = e^{-iH_0E/2}(\mathsf{F}e^{iH_0}\mathsf{F}^\dagger)e^{iH_0E/2}e^{i(\pi/n)WE},$$

on the full Hilbert space. Recall that the fermionic Fourier transform F preserves the Hamming weight. Thus $\mathsf{F}^\dagger$ commutes with $e^{i(\pi/n)WE}$. Commuting the term $e^{i(\pi/n)WE}$ to the left gives

$$\mathsf{C} = e^{-iH_0E/2}\mathsf{F}\Big(e^{iH_0}e^{i(\pi/n)WE}\Big)\mathsf{F}^\dagger e^{iH_0E/2} = V^\dagger e^{iH'}V,$$

where $V = \mathsf{F}^\dagger e^{iH_0E/2}$ and $H' = H_0 + (\pi/n)WE$. Thus $\mathsf{C} = e^{iV^\dagger H'V}$, proving Lemma 7.

5 Conclusion

In this paper, we introduced two ancilla-free circuits implementing the Hidden Weighted Bit function, $O(n^{6.42})$-gate reversible circuit and $O(n^2)$-gate quantum circuit. Our circuits improve best previously known exponential size reversible and quantum ancilla-free circuits into polynomial-size ones. Our results demote **hwb** by removing it from the class of "hard" benchmarks [12]. Our ancilla-free reversible implementation marks a new point in the study of ancilla vs gate count (space-time) tradeoff. Noting a high exponent in the reversible circuit complexity and a more-than-cubic difference between complexities of our best quantum and reversible circuit implementations, we suggest that a further line of inquiry may target improving the reversible implementation. Our work gives an example of a reversible function that can be efficiently implemented by quantum circuits using techniques originally developed in the context of quantum Hamiltonian simulation. It remains to be seen whether **hwb** is an isolated example or there is a larger class of reversible functions that can be efficiently implemented using a similar strategy. More generally, we find it important to study other space-time tradeoffs in quantum computing.

Acknowledgments

The work of Sergey Bravyi and Theodore J. Yoder was supported in part by the IBM Research Frontiers Institute.

REFERENCES

[1] Randal E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 100, no. 8, pp. 677–691, Aug. 1986.

[2] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD-Foundations and Applications*. Berlin, Germany: Springer, 2012.

[3] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 205–213, Feb. 1991.

[4] B. Bollig, M. Löbbing, M. Sauerhoff, andI. Wegener,"On the complexity of the hidden weighted bit function for various BDD models," *RAIRO-Theoretical Inform. Appl.*, vol. 33, no. 2, pp. 103–115, 1999.

[5] D. A. Barrington,"Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1," *J. Comput. Syst. Sci.*, vol. 38, no. 1, pp. 150–164, 1989.

[6] D. Maslov, G. W. Dueck, and D. M. Miller, "Toffoli network synthesis with templates," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 6, pp. 807–817, Jun. 2005.

[7] A. K. Prasad, V. V. Shende, I. L. Markov, J. P. Hayes, andK. N. Patel,"Data structures and algorithms for simplifying reversible circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 4, pp. 277–293, 2006.

[8] Dmitri Maslov, Gerhard W. Dueck, and D. Michael Miller , "Techniques for the synthesis of reversible Toffoli networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 4, pp. 42.1–28, 2007.

[9] J. Donaldand N. K. Jha,"Reversible logic synthesis with Fredkin and Peres gates," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 4, no. 1, pp. 1–19, 2008.

[10] M. Saeedi, M. S. Zamani, M. Sedighi, andZ. Sasanian,"Reversible circuit synthesis using a cycle-based approach," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 6, no. 4, pp. 1–26, 2010.

[11] D. Maslov, G. W. Dueck, and D. M. Miller, "Synthesis of Fredkin-Toffoli reversible networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 6, pp. 765–769, Jun. 2005.

[12] M. Saeediand I. L. Markov,"Synthesis and optimization of reversible circuitsa survey," *ACM Comput. Sur.*, vol. 45, no. 2, pp. 1–34, 2013.

[13] D. V. Zakablukov, "Application of permutation group theory in reversible logic synthesis," 2015, *arXiv:1507.04309*.

[14] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 25, no. 6, pp. 1000–1010, Jun. 2006.

[15] D. Maslov, "Reversible logic synthesis benchmarks page," 2005. [Online]. Available: https://webhome.cs.uvic.ca/~dmaslov/hwbpoly.html

[16] F. Ablayev, A. Gainutdinova, M. Karpinski, C. Moore, and C. Pollett,"On the computational power of probabilistic and quantum branching program," *Inf. Computation*, vol. 203, no. 2, pp. 145–162, 2005.

[17] J. J. Sakurai, *Modern Quantum Mechanics, Revised Edition*. Reading, MA, USA: Addison-Wesley, 1994.

[18] A. Abrikosov, L. Gorkov, and I. Dzyaloshinski, *Methods of Quantum Field Theory in Statistical Physics*. Courier Corporation, 2012. [Online]. Available: https://www.amazon.com/Methods-Quantum-Theory-Statistical-Physics/dp/0486632288

[19] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K. Chan, "Low depth quantum simulation of electronic structure," *Phys. Rev. X*, vol. 8, 2018, Art. no. 011044.

[20] I. D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik , G. K.-L. Chan, and R. Babbush, "Quantum simulation of electronic structure with linear depth and connectivity," *Phys. Rev. Lett.*, vol. 120, no. 11, 2018, Art. no. 110501.

[21] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, vol. 3, pp. 389–400, Sep. 1961.

[22] Ingo Wegener, *The Complexity of Boolean Functions*. Germany: BG Teubner, 1987.

[23] Valerii M. Krapchenko, "Asymptotic estimation of addition time of parallel adder," *Syst. Theory Res.*, vol. 19, pp. 105–122, 1970.

[24] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. 2002. [Online]. Available: https://www.amazon.com/Quantum-Computation-Information-10th-Anniversary/dp/1107002176

[25] F. Verstraete, J. I. Cirac, and J. I. Latorre, "Quantum circuits for strongly correlated quantum systems," *Physical Rev. A*, vol. 79, no 3, 2009, Art. no. 032316.

[26] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Rev. A*, vol. 52, no. 5, 1995, Art. no. 3457.

Sergey Bravyi received the PhD degree in physics from the Landau Institute for Theoretical Physics, Moscow, Russia, in 2003. He is currently a research staff member at IBM Quantum IBM T. J. Watson Research Center. His present research interests include quantum error correction theory and development of quantum algorithms. He has authored more than 60 publications on topics related to quantum computing, information theory, and physics.

Theodore J. Yoder received the PhD degree from the Massachusetts Institute of Technology, in 2018, and has been a research staff member with the quantum computing theory group at IBM T.J. Watson Research since then. He has authored more than 20 publications on quantum physics and quantum computation. His current interests include quantum algorithms and quantum error correction, with focus on novel error-correcting codes and architectural considerations for practical usage.

Dmitri Maslov received the PhD degree, in 2003. He is currently the chief software architect at IBM Quantum (since 2019), IBM's Quantum Computing branch. He provides technical vision and direction for the research and development of software aspects of quantum computing. His research interests include quantum circuits and architectures, quantum compiling, quantum information processing, and reversible logic. The overall goal of Dmitri's research with the establishment of the knowledge base and the development of a set of tools for efficient control and utilization of scalable quantum computers. Before joining IBM, Dmitri was a program director with the Division of Computing and Communication Foundations, Directorate for Computer & Information Science & Engineering, National Science Foundation, Arlington/Alexandria, VA (2008-2018). He directed various programs, including quantum computing and communication, algorithms, computational geometry, complexity, nanocomputing, and symbolic and numeric computing.