QUANTUM BINARY QUADRATIC FORM REDUCTION

* * *

AN $O(N \log N)$ DEPTH CIRCUIT AND APPLICATION TO LATTICE REDUCTION

NICOLAS DAVID[†] AND THOMAS ESPITAU[⁕] AND AKINORI HOSOYAMADA[⁕]

ABSTRACT. Quadratic form reduction enjoys broad uses both in classical and quantum algorithms such as LLL algorithm in lattice reduction. In this paper, we propose the first quantum circuit for definite quadratic form reduction that acheives $\mathrm{O}(n \log n)$ depth, $\mathrm{O}(n^2)$ width and $\mathrm{O}(n^2 \log n)$ quantum gates. The proposed work is based on a binary variant of the reduction algorithm of the definite quadratic form. As side results, we show a quantum circuit performing bit rotation with $\mathrm{O}(\log n)$ depth, $\mathrm{O}(n)$ width, and $\mathrm{O}(n \log n)$ gates, in addition to a circuit performing integer logarithm computation with $\mathrm{O}(\log n)$ depth, $\mathrm{O}(n)$ width, and $\mathrm{O}(n)$ gates.

1. INTRODUCTION

Quantum computing began in the early 1980s when physicist Paul Benioff proposed a quantum mechanical model of the Turing machine. Rejected back then for classical computer, Richard Feynman and Yuri Manin later suggested that a quantum computer had the potential to simulate class of programs that a classical computer could not [1]. Since then, the community started to design algorithms built from this new type of computing relying on the use of so called quantum circuit. Quantum computing appears to be very promising due to its applications to different paradigms among several domains of computer science such as number theory, encryption, search, information theory and more [2] [3] [4], proposing exponential speedup on some specific search tasks. Among these breakthroughs in terms of complexity, the design of efficient reversible shallow circuits for more low level tasks, such as basic integer operations (e.g. addition [5], multiplication [6] and division [7]) is nonetheless a current active domain of research. Based on these base circuits, more complex designs have been made such as quantum circuit for GCD [8]. The goal of this work is to design an effective quantum circuit for reducing quadratic forms. Since computing a GCD can be done by reducing a specific degenerated

[†] INRIA, PARIS, FRANCE.

[⁕] NTT SOCILA INFORMATICS LABORATORIES, NTT COPORATION, TOKYO, JAPAN.

E-mail address: [†]nicolas.david@inria.fr, [⁕]{akinori.hosoyamada.bh, thomas.espitau.ax}@hco.ntt.co.jp.

quadratic form, we seek for a circuit generalizing the one from [8] by reducing general quadratic forms. Because reducing quadratic forms is important as a basic and used computational tool in algorithmic number theory, application to cryptanalysis is also of the utmost important since it gives the first step to the design of a quantum version of the celebrated lattice reduction algorithm LLL. This latter algorithm is crucially used in cryptanalysis and in particular in the security assessment of lattice-based cryptography.

A famous Gauss' reduction algorithm reduces an integer coefficient definite quadratic form with $O(n)$ basic arithmetic operations such as addition, subtraction, and multiplication when coefficients are represented with $O(n)$ bits. Thus, if there are shallow quantum circuits of depth $\mathrm{O}(\log n)$ for basic arithmetic operations, by a straightforward combination of such circuits with Gauss' reduction we achieve a quantum circuit for reduction of depth $\mathrm{O}(n \log n)$. Indeed, some quantum circuits for addition achieve depth $\mathrm{O}(\log n)$ [9, 5, 10]. Moreover, it may be possible to implement asymptotically fast classical algorithms for multiplication such as the ones by Schönhage-Strassen [11] and Harvey-van der Hoeven [12] on a qunatum circuit of $\mathrm{O}(\log n)$. However, usually the constant factors hidden in the order notations for such sophisticated algorithm become quite large.

Our aim is to build a quantum circuit of which depth is small from the view point of not only asymptotic complexity but also exact complexity, so that it can be used for concrete security estimation in a future. To achieve this goal, we seek for a reduction algorithm that does not perform integer multiplications.

Contributions. In this work, the first description of quantum circuits tackling the following problems are proposed:

Computing logarithm's floor of integer in base 2: (i.e. computing the size of the representation of an integer) To the best of our knowledge, this problem does not seem to have been properly tackled in the literature. We give here an optimized shallow circuit to perform this computation, using labeling of a binary tree. Roughly speaking, the leaves will correspond to the values of the bit representation of the input integer and each level of the tree will transmit the information of the size of its children, using a quantum subcircuit for node computation. All in all this gives a $\mathrm{O}(\log n)$-deep, $\mathrm{O}(n)$-wide circuit, where the naive approach (seeking for the left-most 0 in the representation) would give a $\mathrm{O}(n)$-deep circuit.

Computing scalar multiplication by a power 2: We propose an optimized shallow circuit for bit rotation (multiplication by a *variable* power of 2) of depth $\mathrm{O}(\log n)$ and width

Algorithm	Reference	Toffoli depth	$\sharp$ Ancilae	$\sharp$ Toffoli gates
Modular Addition	[5, 10]	$\mathrm{O}(\log n)$	$\mathrm{O}\left(\frac{n}{\log n}\right)$	$\mathrm{O}(n)$
Multiplication	[14]	$\mathrm{O}\left(n^{1.143}\right)$	$\mathrm{O}\left(n^{1.404}\right)$	$\mathrm{O}\left(n^{\log_6 16}\right)$
GCD	[8]	$\mathrm{O}(n \log n)$	$\mathrm{O}(n)$	$\mathrm{O}\left(n^2\right)$
Bit rotation	subsection 4.2	$12 \log n$	$n \log n$	$12n \log n$
Logarithm	subsection 4.3	$4 \log n$	$4n$	$4n$
Quadratic form Reduction	Theorem 5.1	$568n \log n + 896n$	$7n^2 + 26n$	$144n^2 \log n + 2834n^2$

TABLE 1. Summary of reversible shallow circuits.

$\mathrm{O}(n \log n)$ and size $\mathrm{O}(n \log n)$. Our technique relies on the shallow circuit of a rotation by one bit (á la Moore and Nilsson [8]) and combining them with parallel control and fine-grained management of the ancilae.

Reducing definite quadratic forms: Our main circuit is built on a novel generalization of Gauss's reduction, loosely using the binary design of Stein's *Binary* GCD *Algorithm* [13]. Roughly speaking, Stein's algorithm is a classical algorithm to compute GCD which is carefully designed so that it does not perform multiplications and divisions except for those by 2. Since multiplications (and divisions) by 2 can be realized as a simple bit rotation, Stein's algorithm is preferable when we aim to build a GCD circuit of which depth is small not only asymptotically but also from the view point of exact complexity. Indeed, a previous work by Saeedi and Markov [8] builds an efficient quantum circuit for GCD based on this approach. Our strategy is to extend the approach to quadratic form reductions. That is, we first develop a classical algorithm that replaces multiplications by various integers in Gauss' algorithm with computing logarithm and multiplications by a power of 2, and then show how to implement it on a shallow quantum circuit. Using this new (classical) design with the subcircuits introduced above, we achieve a $\mathrm{O}(n \log n)$-deep, $\mathrm{O}\left(n^2\right)$-wide, and $\mathrm{O}\left(n^2 \log n\right)$-size circuit. In fact we analyze the exact complexity of the circuit and a summary can be found in Table 1.

Organisation. First, we will introduce the quadratic form reduction problem together with mathematical tools and elementary quantum gates. Then we will design a global process that solves quadratic reduction and translate each step of this process into quantum circuits. Finally, we will describe the relation between lattice reduction and the LLL algorithm.

2. Preliminaries

In this preliminary section, we introduce the mathematical objects used in this work, namely lattices and quadratic forms, as well as the the elementary quantum gates required to construct the reduction circuit.

2.1. **General notations.**

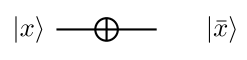
The bold capitals $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ refer as usual to the ring of integers and respectively the field of rational and real. Given a real number x, its integral rounding (by excess) denoted by $\lfloor x \rceil$ returns the integer such that $\lfloor x \rceil - 1/2 < x \leqslant \lfloor x \rceil + 1/2$. The complex conjugation of $z \in \mathbb{C}$ is denoted by the usual bar $\bar{z}$. The logarithm functions are used as $\log$ for the binary logarithm and $\ln$ for the natural one. These operators are extended to operate on vectors and matrices by point-wise composition.

In this paper, n denote a positive integer satisfying $n \geqslant 2$. For a bit b, by $\bar{b}$ we denote its negation. We say an integer x is an n-bit signed (resp., unsigned) integer if $-2^{n-1} \leqslant x < 2^{n-1}$ and x is represented as an n-bit string by two's complement (resp., $0 \leqslant x < 2^n$ and x is represented as an n-bit string). For $x \in \{0,1\}^n$ and $0 \leqslant i < n$, let $x \lll i$ denote the left i-bit rotation on x. For an integer m, let $m \mathbin{\%} n \in \{0, \dots, n-1\}$ denote the remainder of the Euclidean division of m by n (or so-called modulo). For a sequence of integers $i_1, \dots, i_m$ such that $0 \leqslant i_j \leqslant n-1$ for all j and $i_j \neq i_k$ for $j \neq k$, by $(i_1, \dots, i_m)$ we denote the cyclic permutation σ on $\{0, \dots, n-1\}$ such that $\sigma(i_j) = i_{(j+1) \mathbin{\%} n}$ and $\sigma(x) = x$ for $x \in \{0, \dots, n-1\} \setminus \{i_1, \dots, i_m\}$. In particular, (i_1, i_2) denotes the transposition that swaps i_1 and i_2 and let the other index unaltered.

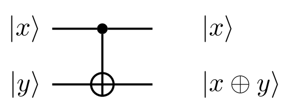
2.2. **Quantum circuit.**

Our quantum circuits are composed of the following quantum gates. Note that Swap and Fredkin gates can be implemented with 3 applications of CNOT and Toffoli gates, respectively.

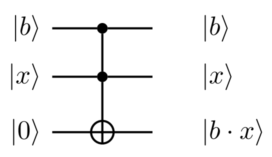
- NOT.



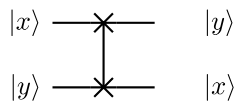
- Bitwise addition (CNOT).



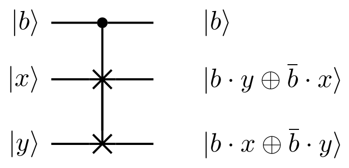
- Toffoli (CCNOT).



- Swap.



- Fredkin (CSWAP).


$|b\rangle$
$|x\rangle$
$|y\rangle$
$|b\rangle$
$|b \cdot y \oplus \bar{b} \cdot x\rangle$
$|b \cdot x \oplus \bar{b} \cdot y\rangle$

In our analysis, we will consider three measures for quantum complexity: Circuit depth, circuit width, and circuit size. Circuit depth corresponds to the execution speed of the quantum circuit. Circuit width corresponds to the number of qubits required. Circuit size is the number of gates in the circuit. We only count Toffoli gates when analyzing depth and size without any order notations, following previous works [9, 5, 10]. For the ease of reading, we will not precise the dimension of the ancilae qubits $|0\rangle$. We ambiguously denote the top and bottom wires of a quantum circuit with n wires by wire 0 and wire $(n-1)$, respectively.

For later use, we show the following lemma.

Lemma 2.1. Let $b \in \{0,1\}$ and $x, y \in \{0,1\}^n$. The operation $|b\rangle\,|x\rangle\,|y\rangle \mapsto |b\rangle\,|x\rangle\,|y \oplus b \cdot x\rangle$ can be implemented with Toffoli depth 1 and n ancillary qubits[1]. The number of Toffoli gates required is n in total.

Proof. This is possible by copying the bit b into n ancilla qubuits, and then applying n Toffoli gates in parallel. $\square$

2.3. **On binary quadratic forms and Gauss' reduction.** We now formally introduce quadratic form and reduced quadratic form.

Definition 2.2. An (integral) quadratic form $\mathcal{Q} = [A, B, C]$ is a polynomial $(A\,X^2 + B\,XY + C\,Y^2) \in \mathbb{Z}[X, Y]$. The integer $\Delta = B^2 - 4AC$ is called the *discriminant* of $\mathcal{Q}$. The form is said to be:

- **Primitive:** when $\gcd(A, B, C) = 1$
- **Degenerate:** when $\Delta = 0$
- **Indefinite:** when $\Delta > 0$
- **Positive (resp. Negative) Definite:** when $\Delta < 0$ and $\mathcal{Q}(x, y) \geqslant 0$ (resp. $\mathcal{Q}(x, y) \leqslant 0$) for any $(x, y) \in \mathbb{R}^2$.

Note that in this work, we will mostly work with definite quadratic forms. A form $\mathcal{Q} = [A, B, C]$ can be represented by the 2×2 matrix $Q = \begin{pmatrix} A & B/2 \\ B/2 & C \end{pmatrix}$, so that the evaluation

[1]The depth can be in $O(\log_2 n)$ even if we count not only Toffoli gates but also all other gates by copying b along with a binary tree. See also Proposition 4.2.

of $\mathcal{Q}$ at x, y is given by $(x, y)Q(x, y)^T$. The modular group $\mathrm{Sl}(2, \mathbb{Z})$[2] have a natural (left) action on the set of quadratic forms given by matrix conjugation, that is defining $S \cdot [A, B, C]$ as $S^T\, Q\, S$. The orbit of a form $\mathcal{Q}$ is called its *class* and the discriminant is an invariant of the class of a form. Two forms lying in the same class are said to be *equivalent*.

Gauss proved that any class contains a *reduced* form in the following sense. Note that $[A, B, C]$ is negative definite if and only if $[-A, -B, -C]$ is positive definite.

Definition 2.3 (Reduced form)**.** A binary quadratic form $[A, B, C]$ is reduced if

$$\left|\sqrt{\Delta} - 2|A|\right| < B < \sqrt{\Delta} \text{ when } [A, B, C] \text{ is indefinite,}$$

$$\left\{\begin{matrix} |B| \leqslant A \leqslant C \\ B \geqslant 0 \text{ if } |B| = A \text{ or } C \end{matrix}\right\} \quad \text{when } [A, B, C] \text{ is positive definite, and}$$

$$[-A, -B, -C] \text{ is reduced when } [A, B, C] \text{ is negative definite.}$$

Gauss' original reduction operates as a sequence of elementary actions of the shape:

$$S = \begin{pmatrix} 0 & 1 \\ 1 & t(\mathcal{Q}) \end{pmatrix}, \qquad \text{where } \mathrm{t}(\mathcal{Q}) = -\mathrm{sgn}(C) \cdot \begin{cases} \left\lfloor \frac{B}{2|C|} \right\rceil & \text{if } |C| \leqslant \sqrt{|\Delta|} \\ \left\lfloor \frac{\sqrt{\Delta}+B}{2|C|} \right\rfloor & \text{if } |C| > \sqrt{|\Delta|} \end{cases},$$

until a reduced form is reached.

3. A binary reduction algorithm for positive definite forms

In this section, we present a reduction algorithm for positive definite forms. The algorithm is designed in such a way that most of its mathematical operations are additions, subtractions, logarithm, or multiplications by a power of 2, so that we can build a shallow quantum circuit based on it. We follow some of the ideas of [8] and consider Stein's algorithm [13] as a starting point. Then we will derive an algorithm for quadratic form reduction more quantizable than the generic algorithm. Note that the algorithm can also be used to reduce negative definite forms: If $[A, B, C]$ is a negative definite form, run the algorithm on $[-A, -B, -C]$ and let $[A', B', C']$ be the output. Then $[-A', -B', -C']$ is the reduced negative definite form in the same class as $[A, B, C]$.

[2]i.e. the group of invertible integer matrices of size 2x2

3.1. **Interlude: Stein's algorithm for gcd.** The greatest common divisor (GCD) of two positive integers A and B can be found by the *Euclidean algorithm* which performs successive division with remainder, given that for $A = Bq + r$ with all positive numbers, $\gcd(A, B) = \gcd(B, r = A\%B)$. This algorithm can be realized as the reduction of the *degenerate* quadratic form $[A^2, 2AC, C^2]$. The *Binary* GCD *Algorithm* [13], also called Stein's algorithm, computes the GCD of two nonnegative integers a and b using subtractions and divisions by two, which are easy to implement in hardware. The algorithm maintains two numbers, starting with a and b, but replaces them at every step with a pair that has the same GCD. The following steps are repeated until either $A = B$ or $A = 0$.

- If $0 \equiv A \equiv B \quad \%2$, then $\gcd(A, B) = 2\gcd(A/2, B/2)$
- If $0 \equiv A \equiv 1 - B \ \%2$, then $\gcd(A, B) = \gcd(A/2, B)$
- If $1 \equiv A \equiv 1 - B \ \%2$, then $\gcd(A, B) = \gcd(A, B/2)$
- If $1 \equiv A \equiv B \quad \%2$, then we ensure that $A \geqslant B$, and use $\gcd(A, B) = \gcd\left(\frac{A-B}{2}, B\right)$

Compare to the classical Euclidean algorithm it avoids to perform divisions of integers and only deals with addition and shifts. For n-bit integers, each step takes linear time, Thus, the binary GCD algorithm needs $O(n^2)$ time so that it does not improve asymptotic performance compared to Euclidean algorithm or its variants by Lehmer [15, Chapter 4.5.3 Theorem E]. However, on average, it uses 60% fewer bit operations than the Euclidean algorithm as reported by [16]. Remark that we can factorize some divisions by 2 by directly dividing by the greatest power of 2 dividing both A and B. It now makes appearing that we are in substance reconstructing the binary decomposition of the greatest common divisor.

3.2. **Towards a binary reduction.** We now turn to the devise of a Stein' like binary reduction for quadratic forms. Likewise, it aims at avoiding the divisions appearing in Gauss' reduction. This core construction is quite similar to the usual reduction algorithm, but instead of acting on the form with transvections matrices of the form $S_m = \begin{pmatrix} 0 & 1 \\ 1 & m \end{pmatrix}$ for arbitrary $m \in \mathbb{Z}$, we only restrict ourselves to transvections with coefficients power-of-two. Recall that given a matrix $G = \begin{pmatrix} A & B/2 \\ B/2 & C \end{pmatrix} \in M(2, \mathbb{R})$, then we have the congruence:

$$S_m^t \cdot G \cdot S_m = \begin{pmatrix} C & B/2 + mC \\ B/2 + mC & A + mB + m^2C \end{pmatrix} \tag{1}$$

For m being a power-of-two $m \cdot a$ can be computed by shifting the bits of a. Hence, $S_m^t \cdot Q \cdot S_m$ can be computed using only shifts and additions. Therefore, we can now use the same principle as for Stein's algorithm: in order to circumvent the transvections requiring multiplications and divisions, we are instead implicitly following the corresponding binary decomposition and apply sequences of bits operations.

Algorithm 1 — Positive Definite Reduction

```
Input  : $\mathcal{Q}$ positive definite quadratic form represented by $[A, B, C]$
Output : A reduced form in $[\mathcal{Q}]$

m ← 0, $\varepsilon \leftarrow \mathrm{sgn}(B)$
if $C < A$ then $(C, A) \leftarrow (A, C)$
if $\neg(|B| \leqslant 2A)$ then $m \leftarrow 2^{\lfloor \log_2 |B| \rfloor - \lfloor \log_2 A \rfloor - 1}$
else $m \leftarrow \left\lfloor \frac{|B|}{2A} \right\rceil$
if $m = 0$ then return $[A, (-1)^{\delta(A=-B)} B, C]$
else Reduce$(C - \varepsilon m B + m^2 A, B - \varepsilon 2 m A, A)$
```

Theorem 3.1. Given $\mathcal{Q} = [A, B, C]$ a quadratic binary form where $|A|, |B|, |C| < 2^n$, then the algorithm returns a reduced form by making at most n recursive calls.

Proof. First of all, note that positive definite forms always satisfy $A, C > 0$.

Correctness: First, we will assume termination to prove correctness, then we will study termination and complexity. Remark that each operation of the algorithm on the quadratic form can be represented as in (1) and thus preserves the class of the form:

- the swap of A and C on line 2 is realized by the action of S_0.
- the operation $[A, B, C] \mapsto [A, (-1)^{\delta(A=-B)} B, C]$ on line 5 can be realized by the action of $(S_0 \cdot S_1)$.
- the transformation $[A, B, C] \mapsto [C, -\varepsilon m B + A^2, B - \varepsilon 2 m A, A]$ of line 6 can be realized by the action of $(S_0 \cdot S_{-\varepsilon m} \cdot S_0)$.

By a direct induction, we never leave the orbit of the form and in particular the output is in the orbit of the input. Assuming termination, we can claim that condition on line 5 is true and that the algorithm crosses line 2. Hence both of the following conditions

$$m = 0 \qquad (2) \qquad\qquad C \geqslant A \qquad (3)$$

hold. Since the function $x \mapsto 2^x$ is a strictly positive function, (2) leads to line 3 condition being false. Therefore we have $m = \left\lfloor \frac{|B|}{2A} \right\rceil = 0$, which implies that $|B| \leqslant A$. Using (3), we can state $|B| \leqslant A \leqslant C$ In addition, if $|B| = A$, then the output always satisfies $B = A$ due to the coefficient $\delta(A = -B)$ on line 5. Hence, the quadratic form is reduced.

Termination: Denote by A_i, B_i, C_i the values taken respectively by the variables A, B, C at the i-th recursive call. We will show that the sequence of non-negative integers $|B_1|, |B_2|, \ldots$ satisfies that $|B_{i+1}| < |B_i|$ for all i. Without loss of generality, we can assume $A_i < C_i$ (due to line 2 of the algorithm).

- If m is set on line 3, then $m = 2^{\lfloor \log_2 |B| \rfloor - \lfloor \log_2 A \rfloor - 1}$. By remarking $0 \leqslant x - \lfloor x \rfloor \leqslant 1$, we can state :

$$\begin{aligned}
|B_{i+1}| &= |B_i - 2\varepsilon m A_i| \\
&= \left| \varepsilon |B_i| - \varepsilon 2^{\lfloor \log_2 |B_i| \rfloor + \log_2 A_i - \lfloor \log_2 A_i \rfloor} \right| \\
&= |B_i| \Big(1 - 2^{(\log_2 A_i - \lfloor \log_2 A_i \rfloor) - (\log_2 |B_i| - \lfloor \log_2 |B_i| \rfloor)} \Big) \\
&\leqslant |B_i|(1 - 1/2) = |B_i|/2
\end{aligned}$$

- If m is set on line 4, then $m = \left\lfloor \frac{|B_i|}{2A_i} \right\rceil$ and condition line 3 is false. Therefore $|B_i| \leqslant 2A_i$ holds, which implies that m is 0 or 1. If $m = 0$, then the algorithm terminates. Thus we will study the case $m = 1$. Note that $m = \left\lfloor \frac{|B_i|}{2A_i} \right\rceil = 1$ implies $A_i < |B_i| \leqslant 2A_i$. Hence $|B_{i+1}| = \big|B_i - 2\varepsilon m A_i\big| = \big||B_i| - 2A_i\big| = 2A_i - |B_i| \leqslant |A_i| < |B_i|$ holds.

Complexity: We first show that, if m is set to be 1 at line 4 of the ith recursive call, the algorithm will terminate at the $(i+1)$st recursive call. Again, without loss of generality we assume $A_i \leqslant C_i$.

- Step i. In the arguments to prove correctness and termination, we showed the followings hold when m is set to be 1 at line 4:

$$A_i < |B_i| \leqslant 2A_i, |B_{i+1}| = 2A_i - |B_i|, C_{i+1} = A_i, A_{i+1} = C_i - |B_i| + A_i.$$

In particular, we can deduce that

$$C_i - A_i \leqslant 2(C_i + A_i - |B_i|) \tag{4}$$

holds.

- Step $i+1$. First, we can easily confirm that $|B_{i+1}| \leqslant A_{i+1}, 2C_{i+1}$ holds. In particular, m is set to be 0 or 1 at line 4. If $A_{i+1} \geqslant C_{i+1} = A_i$, line 2 swaps A_{i+1} and C_{i+1}. Hence

$$m = \left\lfloor \frac{|B_{i+1}|}{2A_i} \right\rceil = \left\lfloor 1 - \frac{|B_i|}{2A_i} \right\rceil = 0 \tag{5}$$

holds, where we used the condition $A_i < |B_i|$ for the last equality. If $A_{i+1} \leqslant C_{i+1} = A_i$, then

$$m = \left\lfloor \frac{|B_{i+1}|}{2A_{i+1}} \right\rceil = \left\lfloor \frac{2A_i - |B_i|}{2(C_i + A_i - |B_i|)} \right\rceil = \left\lfloor \frac{1}{2} - \frac{C_i - A_i}{2(C_i + A_i - |B_i|)} \right\rceil = 0,$$

follows from (4). Therefore the algorithm stops at the $(i+1)$st recursive call.

Note that every time m is set at line 3, the quantity $|B_i|$ gets reduced in half. Since $|B_i| < 2^n$, the operation of line 3 is not executed more than $(n-1)$ times. Moreover, line 4 cannot happen more than 2 times in a row, allowing to conclude that the algorithm does at most n recursive calls.

□

4. Quantum Circuits for Elementary Operations

Before diving in the precise complexity analysis of the circuit corresponding to the reduction algorithm presented in Section 3, we introduce and analyse the main gadgets used in the design: shallow quantum circuits for addition (in addition to subtraction and absolute value), bit rotation, and logarithm.

4.1. **Quantum circuit for addition, subtraction, and absolute value.**

4.1.1. *Addition.* To build a shallow quantum circuits for reductions of quadratic forms, we need a quantum circuit for addition of which depth, width, and size are in $O(\log n)$, $O(n)$, and $O(n)$, respectively (see also Figure 1). Such a shallow quantum circuit is first shown by Draper et al. [9] and later improved by Takahashi and Kunihiro [5], and Takahashi et al. [10]. Each of them meets our demands and is composed of only NOT, CNOT, and Toffoli gates [3] and thus could also be implemented on a classical computer. For instance, the adder for n-bit integers in [5] can be implemented with Toffoli depth $30 \log n$, $(3n/\log n)$ ancillary qubits, and $29n$ Toffoli gates in total.

[3]In fact we need a circuit for *modular* addition rather than usual addition and some of the previous works implement the latter. Still, we can easily convert a quantum circuit for addition into one for modular addition without changing complexity.

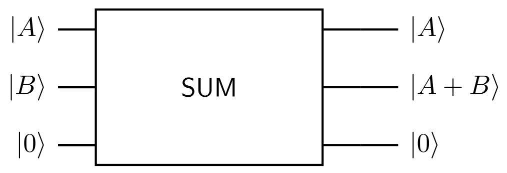


FIGURE 1. Quantum circuit for addition.

Quantum circuits to compute subtraction and absolute value can be realized with the same asymptotic complexity, which we elaborate below.

4.1.2. *Subtraction.* Subtractions can be computed by uncomputing additions. Hence the circuit of subtraction for n-bit integers can be implemented with Toffoli depth $30 \log n$, $(n + 3n/\log n)$ ancillary qubits, and $29n$ Toffoli gates in total. In what follows, we assume $-x$ is computed from x by subtracting x from 0.

4.1.3. *Absolute value.* Here we explain how we implement the operation $|x\rangle |y\rangle \mapsto |x\rangle |y \oplus |x|\rangle$ for n-bit integers x and y on a quantum circuit. Note that we can tell whether a number x is negative by checking if $b_{\text{msb}} = 1$, where b_{msb} is the most significant bit of x. We compute the absolute value $|x|$ as follows.

(i) Compute $-x$ and copy b_{msb} to ancillary qubits to obtain $|x\rangle |-x\rangle |b_{\text{msb}}\rangle |y\rangle$.
(ii) Copy and write $-x$ to the output register iff $b_{\text{msb}} = 1$ to obtain $|x\rangle |-x\rangle |b_{\text{msb}}\rangle |b_{\text{msb}} \cdot (-x) \oplus y\rangle$.
(iii) Copy and write x to the output register iff $b_{\text{msb}} = 0$ by flipping b_{msb} and sequentially applying n CNOT gates on $\bar{b}_{\text{msb}}$ and each bit of x to obtain $|x\rangle |-x\rangle |\bar{b}_{\text{msb}}\rangle |b_{\text{msb}} \cdot (-x) \oplus \bar{b}_{\text{msb}} \cdot x \oplus y\rangle$.
(iv) Uncompute (i) to obtain $|x\rangle |0\rangle |b_{\text{msb}} \cdot (-x) \oplus \bar{b}_{\text{msb}} \cdot x \oplus y\rangle = |x\rangle |0\rangle ||x| \oplus y\rangle$.

(i) requires depth $30 \log n$, $(n + (3n/\log n) + 1)$ ancillary qubits, and $29n$ Toffoli gates in total. By Lemma 2.1, (ii) and (iii) can be done by additional Toffoli depth 2 and $2n$ Toffoli gates in total. (Note that the output register is the rightmost one.) (iv) requires additional Toffoli depth of $30 \log n$ and $29n$ Toffoli gates in total. In summary, the circuit to compute absolute values for n-bit integers can be implemented with depth $(60 \log n + 2)$, $(n + (3n/\log n) + 1)$ ancillary qubits, and $60n$ Toffoli gates in total.

4.2. **Quantum circuit for bit rotation.** We now turn to the design of an efficient circuit for (qu)bit rotation. More formally, given a parameter $i \in \mathbb{Z}$ and a set of n qubits, the (left) rotation of amount i is the application of the unitary operator encoding the permutation on $\{0, \ldots, n-1\}$

that maps j to $(j - i) \,\%\, n$:

$$\begin{pmatrix} 0 & \cdots & i & i+1 & \cdots & n-1 \\ n-i & \cdots & 0 & 1 & \cdots & n-i-1 \end{pmatrix}$$

We insist on the fact that the shift amount is not fixed and is encoded in the input of the circuit. Hence the circuit specifies as in Figure 2. In what follows, we build a circuit for rotation of which depth, width, and size are $O(\log n)$, $O(n \log n)$, and $O(n \log n)$, respectively.

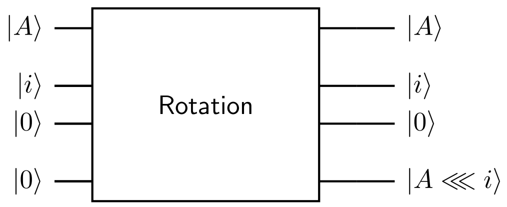


FIGURE 2. The quantum circuit for bit rotation.

We first study the simplified case where $i = 2^m$ or 0 for a fixed $m \in \{0, \ldots, \log_2 n - 1\}$. We denote the quantum circuit for this simplified case by S_m (see Figure 3). We will realize a quantum circuit for bit rotation by combining S_m for $m = 0, \ldots, \log_2 n - 1$.

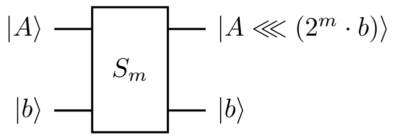


FIGURE 3. The quantum circuit S_m ($A \in \{0,1\}^n$ and $b \in \{0,1\}$).

4.2.1. *How to realize S_0 as a shallow circuit.* The naive approach for 1-bit rotation consists by swapping iteratively each bits by one position starting from the last one, resulting in a circuit of depth $O(n)$, as depicted in the left circuit below. However, apparently this approach does not lead to our goal of an $O(\log n)$ depth circuit. Therefore we follow the idea from the previous work on GCD computation by Saeedi and Markov [8], which use the following two results from [17] to implement S_0 on a circuit of depth $O(\log n)$.

Proposition 4.1 (Proposition 1 in [17]). The cyclic permutation on n qubits that moves the qubit at wire i to wire $(i - 1)$ $(i = 0, \ldots, n - 1)$ can be implemented on a quantum circuit of which depth and size are in $O(1)$ and $O(n)$ without any ancillae by first swapping wire $(i + 1) \,\%\, n$ and

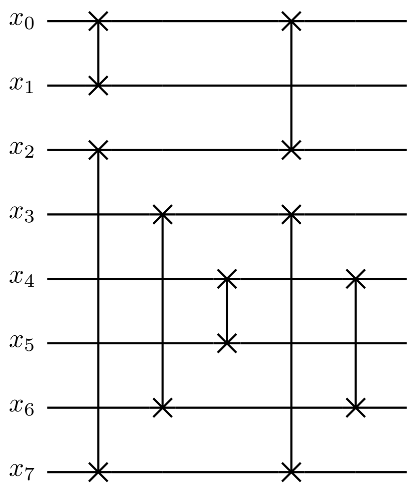


FIGURE 4. Example of constant time 1-bit rotation for 8 qubits. The left four gates and the right three gates can be applied at once in parallel. (This figure is based on Figure 5 of [8].)

wire $(n-i)$ % n for $i = 0, \dots, n/2-1$ at once in parallel, and then swapping wire $(i+1)$ % n and wire $(n-1-i)$ % n at once in parallel for $i = 0, \dots, n/2-2$.

Proposition 4.2 (Proposition 2 in [17])**.** Let $U_1, \dots, U_n$ be quantum gates (or, unitary operators that can be implemented with $O(1)$ gates), and CU_i be the controlled-U_i gate. Then, the series of n controlled gates $CU_1, \dots, CU_n$ such that the controll qubits of CU_i and CU_j are the same but the target qubits are disjoint for all $i \neq j$ can be parallelized to $O(\log n)$ depth by using $O(n)$ ancillae and $O(n)$ gates are used.

Proposition 4.1 holds because the permutation on the set $\{0, \dots, n-1\}$ that maps i to $(i-1)$ % n can be decomposed into the product of transpositions $\left((1,0)\prod_{i=1}^{n/2-1}(i+1, n-i)\right) \cdot \left(\prod_{i=0}^{n/2-2}(i+1, n-i-1)\right)$. An example on 8 qubits is given in Figure 4. See [17] for details.

The parallelization of Proposition 4.2 can be realized by first copying the control qubit to n ancillary qubits (which can be done in depth $O(\log n)$ with $O(n)$ CNOT gates), second applying $CU_1, \dots, CU_n$ at once in parallel (the i-th ancillary qubit is used as the control qubit of CU_i), and finally uncomputing the copies of the control qubit. See also Figure 5.

Saeedi and Markov observed that Proposition 4.2 enables us to replace the swap gates in Proposition 4.1 with Fredkin gates that swap wires if and only if $b = 1$ while keeping the depth

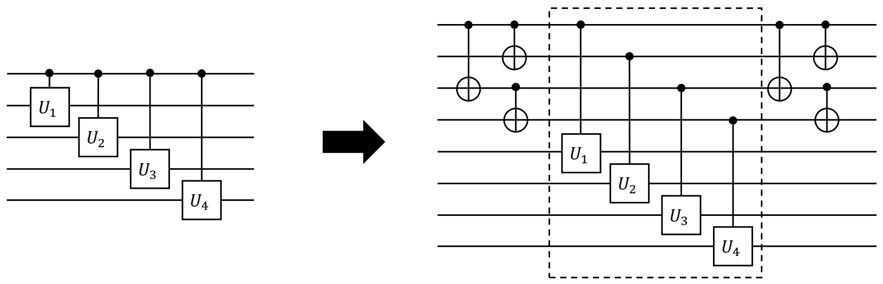


FIGURE 5. Parallelization of 4 controlled unitary gates. The gates in the dotted rectangle are applied at once in parallel. (This figure is based on Figure 5 of [8].)

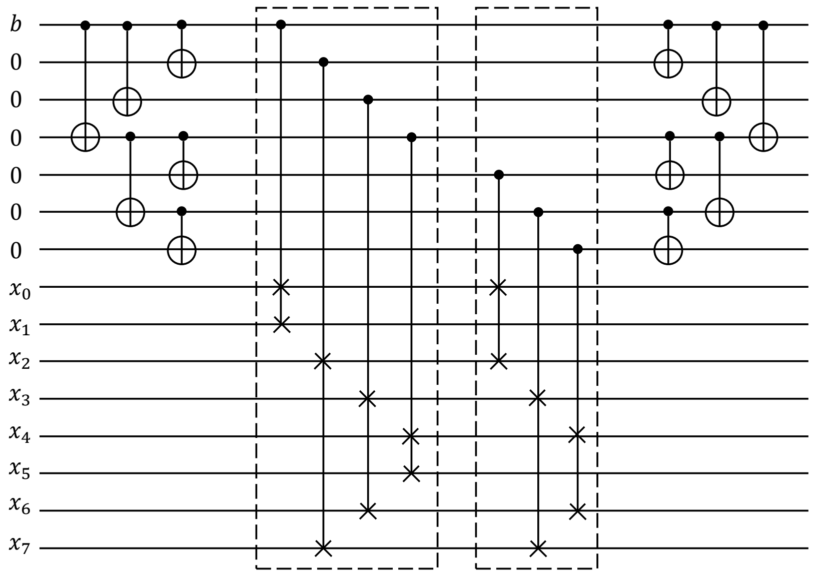


FIGURE 6. Implementation of S_0 for $n = 8$. Fredkin gates surrounded by a dotted rectangle are applied at once in parallel.

of the circuit in $O(\log n)$, which implies that we can implement the controlled 1-bit rotation (i.e., S_0) on a quantum circuit of width $O(n)$, depth $O(\log n)$, and size $O(n)$. See also Figure 6 for the example when $n = 8$.

4.2.2. *How to realize S_m as a shallow circuit for $m > 0$.* Proposition 4.1 implies that *any* fixed permutation on n wires can be implemented on a quantum circuit of depth $O(1)$ since arbitrary permutation can be uniquely decomposed into a product of cyclic permutations on disjoint subsets[4]. In particular, we can decompose the permutation σ_m on the set $\{0, \dots, n-1\}$ such that $\sigma_m(i) = (i - 2^m) \,\%\, n$ into the product of 2^m cyclic permutations as $\sigma_m = \prod_{i=0}^{2^m-1} \sigma_m^{(i)}$, where $\sigma_i^{(i)} := (i, (i-2^m) \,\%\, n, \dots, (i - (n/2^m - 1) \cdot 2^m) \% \, n)$. We implement S_m for $m > 0$ on a quantum circuit of depth $O(\log n)$, width $O(n)$, and size $O(n)$ in the same way as we do for S_0: We first implement σ_m on a circuit of depth $O(1)$ and width $O(n)$ with swap gates by applying Proposition 4.1 to all $\sigma_m^{(i)}$, and then replace the swap gates with Fredkin gates that swap wires if and only if $b = 1$ while keeping the depth in $O(\log n)$, by using Proposition 4.2.

4.2.3. *How to implement the circuit for bit rotation: First attempt.* To perform i-bit rotation for any parameter i, we consider its decomposition in base 2 say $i = \sum_{m=0}^{\log_2 n - 1} i_m 2^m$. The i-bit rotation can be realized by applying 2^m-bit rotation if and only if $i_m = 1$ for $m = 0, \dots, \log_2 n - 1$ in a sequential order. Given an input $|i\rangle = |i_0 \cdots i_{\log_2 n - 1}\rangle$ and $|A\rangle$ ($i \in \{0,1\}^{\log_2 n}$ and $A \in \{0,1\}^n$), our circuit runs as follows: First, apply S_m to $|A\rangle |i_m\rangle$ in a sequential order for $m = 0, \dots, \log_2 n - 1$ to obtain the state $|A \lll i\rangle |i\rangle$. Second, copy the value $(A \lll i)$ to ancillary qubits to obtain the state $|A \lll i\rangle |i\rangle |A \lll i\rangle$. Finally, apply uncompute S_m for $m = \log_2 n, \dots, 0$ to obtain the final state $|A\rangle |i\rangle |A \lll i\rangle$. See also Figure 7.

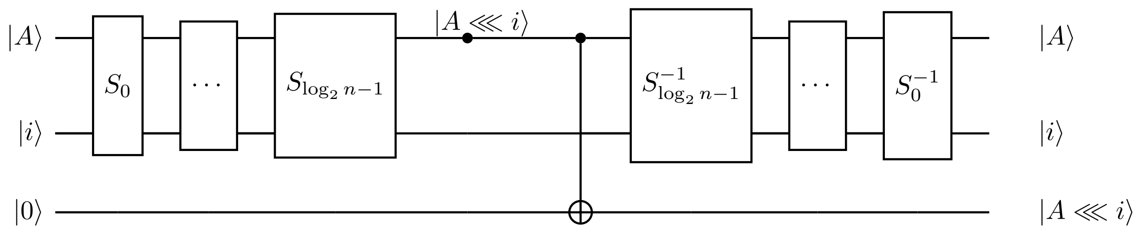


FIGURE 7. Implementation of rotation for any i with $O((\log n)^2)$ depth. S_m involves the qubit $|i_j\rangle$ if and only if $j = m$.

4.2.4. *Reducing the depth from $O\big((\log n)^2\big)$ to $O(\log n)$.* Here we show how to further reduce the depth of the quantum circuit in Figure 7. The reason that each S_m requires depth $O(\log n)$ (but not $O(1)$) is that we have to copy the control qubit i_m into n ancillary qubits. The depth of the entire circuit of Figure 7 becomes $\mathrm{O}\big((\log n)^2\big)$ because this copy operation is performed for each $m \in \{0, \dots, \log_2 n - 1\}$. However, we observe that the control qubits for S_m and $S_{m'}$ are distinct for $m \neq m'$. Therefore we can reduce the depth of the circuit from $\mathrm{O}\big((\log n)^2\big)$ to $\mathrm{O}(\log n)$ by

[4]In fact the original version of Proposition 4.1 in [17] states that any permutation can be implemented on a quantum circuit of depth $O(1)$. We stated the proposition like above for convenience of explanation.

performing the copy operations on the control qubits $i_0, \ldots, i_{\log_2 n-1}$ at once in parallel for all m at the beginning of the circuit (and at the end of the circuit for uncomputations) with $O(n \log n)$ ancillae. Eventually, we obtain a quantum circuit for bit rotation of which depth, width, and size are $O(\log n)$, $O(n \log n)$, and $O(n \log n)$, respectively.

4.2.5. *Exact complexity without order symbols.* So far we have provided only asymptotic complexity, but it is straightforward to check that the exact (Toffoli) depth, ancillary qubits, and (Toffoli) size of the above circuit (when the A is an n-bit unsigned integer and $i \in \{0,1\}^{\lceil \log_2 n \rceil}$) are at most $12\lceil \log n \rceil$, $n\lceil \log n \rceil$, and $12n\lceil \log n \rceil$. (Note that the Fredkin gate can be implemented by applying the Toffoli gate three times.)

4.3. **Quantum circuit for logarithm.** This section shows a quantum circuit to compute $\lfloor \log_2 B \rfloor$ for an unsigned n-bit integer $B \geqslant 0$ of which depth, width, and size are $O(\log n)$, $O(n)$, and $O(n)$, respectively (See also Figure 8). In what follows, we first show a quantum circuit for $B \geqslant 0$ and B is an n-bit unsigned integer. If $B = 0$, we define $\log_2 B := 0$ for convenience.

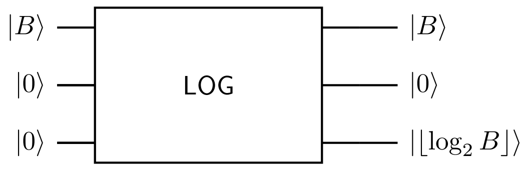


FIGURE 8. Quantum circuit for logarithm.

Our idea to realize is to utilize labeling of a binary tree. Let us denote bit i of $B \in \{0,1\}^n$ by b_i ($B = b_0 b_1 \cdots b_{n-1}$). Let B_L and B_R be the most and least significant $n/2$ bits of B, respectively.

First, we observe that $\lfloor \log_2 B \rfloor$ can be computed from $\lfloor \log_2 B_L \rfloor$ and $\lfloor \log_2 B_R \rfloor$ as follows when $B > 0$.

$$\text{If } B_L \neq 0, \lfloor \log_2 B \rfloor = \lfloor \log_2 B_L \rfloor + n/2. \text{ If } B_L = 0, \lfloor \log_2 B \rfloor = \lfloor \log_2 B_R \rfloor. \tag{6}$$

From this observation, we notice that $\lfloor \log_2 B \rfloor$ can be computed by labeling the nodes of the complete binary tree of depth $\log_2 n$, where the label of a node v is determined as follows:

(a) If v is the i-th leaf, v is labeled with (b_i, ε) (the leftmost and rightmost leaves are labeled with (b_0, ε) and (b_{n-1}, ε), respectively). Here, ε denotes the empty string.
(b) If v is not a leaf, let $(b_i, \varepsilon), (b_{i+1}, \varepsilon), \ldots, (b_{i+j}, \varepsilon)$ denote the labels of the leaves that are descendants of v, and define $B(v) := b_i \cdots b_{i+j}$. If $B(v) \neq 0$, label v with $(1, \lfloor \log_2 B(v) \rfloor)$. If $B(v) = 0$, label v with $(0, 0)$.

Note that, if we denote the left and right children of a node v by v_L and v_R, then $B(v) = B(v_L)||B(v_R)$ holds. In particular, we have $B(\text{root}) = B$. See also Figure 9.

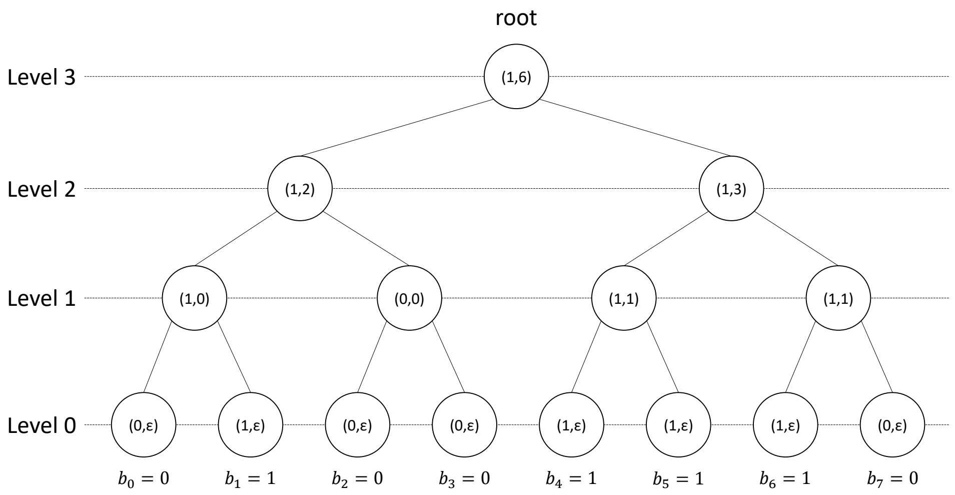


FIGURE 9. Labels of the nodes when $n = 8$ and $B = 01001110$ ($\lfloor \log_2 B \rfloor = 6$).

Since the equation (6) holds, we can compute the label of a node v once the labels of their children v_L and v_R are computed. More precisely, suppose v is a node at level j. Let $\mathsf{label1}(v) \in \{0,1\}$ and $\mathsf{label2}(v) \in \{0,1\}^j$ denote the first and second entries of the label of v, respectively. Then

$$\mathsf{label1}(v) = \mathsf{label1}(v_L) \vee \mathsf{label1}(v_R) \tag{7}$$

holds. In addition, $\mathsf{label2}(v)$ is computed as $\mathsf{label2}(v) = \mathsf{label2}(v_L) + 2^{j-1} = 1||\mathsf{label2}(v_L)$ if $\mathsf{label1}(v_L) = 1$ and $\mathsf{label2}(v) = 0||\mathsf{label2}(v_R)$ if $\mathsf{label1}(v_L) = 0$. In particular, the followings hold.

$$\mathsf{msb}(\mathsf{label2}(v)) = \mathsf{label1}(v_L), \tag{8}$$

$$\mathsf{i\text{-}msb}(\mathsf{label2}(v)) = \begin{cases} (\mathsf{i}-1)\text{-}\mathsf{msb}(\mathsf{label2}(v_L)) \text{ if } \mathsf{label1}(v) = 1, \\ (\mathsf{i}-1)\text{-}\mathsf{msb}(\mathsf{label2}(v_R)) \text{ if } \mathsf{label1}(v) = 0. \end{cases} \tag{9}$$

Here, msb denotes the most significant bit, and i-msb denotes the i-th most significant bit.

So far we assumed $B > 0$ but the arguments extend to the case $B = 0$, by setting all the labels as $(0,0)$ except for leaves and $(0,\varepsilon)$ for all the leaves. By using (7) - (9), we can compute all the

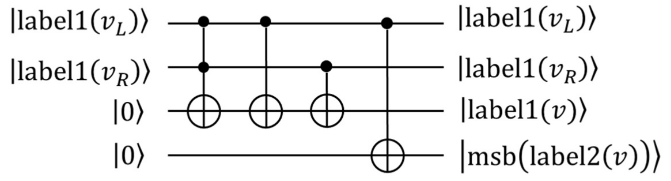

$|\text{label1}(v_L)\rangle$
$|\text{label1}(v_R)\rangle$
$|0\rangle$
$|0\rangle$
$|\text{label1}(v_L)\rangle$
$|\text{label1}(v_R)\rangle$
$|\text{label1}(v)\rangle$
$|\text{msb}(\text{label2}(v))\rangle$

FIGURE 10. The quantum circuit to compute $\mathsf{label1}(v)$ and $\mathsf{msb}(\mathsf{label2}(v))$.

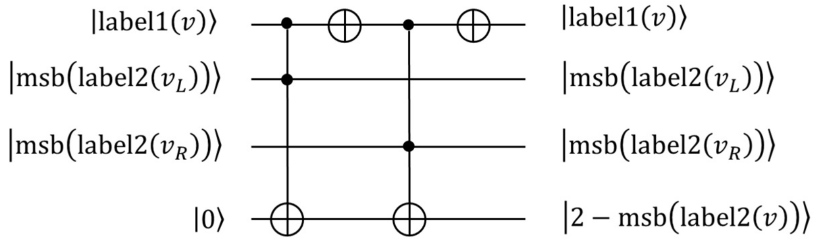

$|\text{label1}(v)\rangle$
$|\text{msb}(\text{label2}(v_L))\rangle$
$|\text{msb}(\text{label2}(v_R))\rangle$
$|0\rangle$
$|\text{label1}(v)\rangle$
$|\text{msb}(\text{label2}(v_L))\rangle$
$|\text{msb}(\text{label2}(v_R))\rangle$
$|2-\text{msb}(\text{label2}(v))\rangle$

FIGURE 11. The quantum circuit to compute 2-$\mathsf{msb}(\mathsf{label2}(v))$.

labels on a quantum circuit as follows.

1. Compute $\mathsf{label1}(v)$ and $\mathsf{msb}(\mathsf{label2}(v))$ for all the nodes v at level j parallelly and simultaneously by using (7) and (8) from the values $\{\mathsf{label1}(v')\}_{v':\text{level } j-1}$, for each $j = 1, 2, \ldots, \log_2 n$ in a sequential order. The quantum circuit for this procedure is shown in Fig. 10. At this moment, computations for all the labels of node v of level 0 and level 1 are completed, but the labels of level j for $j \geqslant 2$ are incomplete.
2. For all the nodes v at level $j \geqslant 2$, compute 2-$\mathsf{msb}(\mathsf{label2}(v))$ parallelly and simultaneously, by using (9). This can be done by just copying the bit $\mathsf{msb}(\mathsf{label2}(v_L))$ if $\mathsf{label1}(v) = 1$, or $\mathsf{msb}(\mathsf{label2}(v_R))$ if $\mathsf{label1}(v) = 0$, into a new register (See also Fig. 11). At this moment, computations for the labels of level 2 is completed.
4. Copy $\mathsf{label2}(\text{root}) = \lfloor \log_2 B \rfloor$ into the output register.
5. Uncompute Steps 1-3.

4.3.1. *Complexity analysis.*

- Depth. In Step 1, the Toffoli depth of the computation for level j is 1 for each $j = 1, \ldots, \log_2 n$. Thus the Toffoli depth of Step 1 is requires $\log_2 n$. Similarly, in Steps 2 and 3, the computation of j-$\mathsf{msb}(\mathsf{label2}(v))$ is done with Toffoli depth 2 for each $j = 2, \ldots, \log_2 n$. Hence Steps 2 and 3 require $2(\log_2 n - 1)$ depth. Depth for Step 4 is does not use Toffoli gates. Thus the overall depth is at most $2 \times (\log_2 n + 2 \times (\log_2 n - 1) \leqslant 4 \log_2 n$. Hence the overall Toffoli depth is $4 \log_2 n$. (Even if we count not only Toffoli gates but also other gates, a similar analysis shows that the depth is in $O(\log_2 n)$)

- Width. Let $h := \log_2 n$. We need $(j+1)$ ancillary qubits to compute and store the label for each node of level j, and there are 2^{h-j} nodes exist at level j. Hence the width of the whole circuit is at most $\sum_{j=0}^{h}(j+1) \cdot 2^{h-j}$. We have
$$\sum_{j=0}^{h}(j+1) \cdot 2^{h-j} = 2^h \sum_{j=1}^{h} j \cdot 2^{-(j-1)} = 2^h \sum_{j=1}^{h} j \cdot x^{(j-1)} \Bigg|_{(x=2^{-1})}$$
and
$$\sum_{j=1}^{h} j \cdot x^{(j-1)} = \left(\frac{x^{h+1}-1}{x-1} \right)' = \frac{(h+1)x^h(x-1) - (x^{h+1}-1)}{(x-1)^2}.$$
Hence the total number of ancillary qubits required is at most
$$2^h \frac{(h+1)(-2^{-(h+1)}) - (2^{-(h+1)}-1)}{(-1/2)^2} \leqslant 2^{h+2} = 4n.$$
- Size. In Step 1, the number of Toffoli gates required for the computation for level j is $2^{\log_2 n - j}$ for each $j = 1, \ldots, \log_2 n$. Thus the total number of Toffoli gates required for Step 1 is $\sum_{j=1}^{\log_2 n} 2^{\log_2 n - j} = \sum_{j=0}^{\log_2 n - 1} 2^j \leqslant 2^{\log_2 n} = n$. Similarly, in Steps 2 and 3, the computation of $\mathsf{j\text{-}msb}(\mathsf{label2}(v))$ uses $2 \cdot 2^{\log_2 n - j}$ Toffoli gates for each $j = 2, \ldots, \log_2 n$. Hence Steps 2 and 3 require $\sum_{j=2}^{\log_2 n} 2 \cdot 2^{\log_2 n - j} = 2 \sum_{j=0}^{j=\log_2 n - 2} 2^j \leqslant n$ Toffoli gates. Depth for Step 4 is does not use Toffoli gates. Thus the overall size of the circuit is at most $2(n+n) = 4n$. (Even if we count not only Toffoli gates but also other gates, a similar analysis shows that the size is in $O(n)$)

So far we have assumed n is a power of two. Even if n is not a power of 2, the above arguments work by considering a tree of height $\lceil \log_2 n \rceil$. In particular, $\lfloor \log_2 B \rfloor$ can be computed with Toffoli depth $8 \log_2 n$, $8n$ ancillary qubits, and $8n$ Toffoli gates in total (note that $2^{\lceil \log_2 n \rceil} \leqslant 2n$).

5. Quantum Circuit for Reduction on Binary Quadratic Forms

This section shows how to implement Algorithm 1 on a shallow quantum circuit by using the circuits introduced in Section 4 and analyze its complexity. The depth, width, and size of the resulting quantum circuit is $O(n \log n)$, $O(n^2)$, and $O(n^2 \log n)$, respectively. In the following, we suppose that we are given a positive definite quadratic form $[A, B, C]$ represented by three n-bit signed integers. Without loss of generality (up to padding by the appropriate amount of zeros) we shall suppose that n is a power of two (the asymptotic depth of the circuit remains unchanged by such padding, but the analysis is slightly simpler in this case).

5.1. **An Alternative Description of Algorithm 1.** The description of Algorithm 1 is simple but not favorable as a base for a quantum circuit because it is a recursive algorithm of which running time depends on inputs, whereas the running time of quantum circuits is independent of inputs. Thus, first we rewrite Algorithm 1 as in Algorithm 2 so that the running time will not depend on inputs. The behavior of Algorithm 2 is essentially the same as that of Algorithm 1 and Algorithm 2 certainly outputs the reduced form because the followings properties hold on Algorithm 1, as we showed in the proof of Theorem 3.1.

- Every time m is set on line 3, $|B|$ is reduced at least half. Hence m is set on line 3 at most n times.
- If m is set on line 5, m is 0 or 1. If $m = 0$, then the algorithm immediately stops, outputting the result. If $m = 1$, then in the next recursive call m is set to be 0 on line 5 and the algorithm terminates.

Roughly speaking, line 3-13, line 14-18, and line 19-21 of Algorithm 2 corresponds to the cases in Algorithm 1 when m is set on line 3, m is set to be 1 on line 4, and m is set to be 0 on line 4, respectively.

Algorithm 2 — Positive Definite Reduction (an alternative description)

```
Input  : $\mathcal{Q}$ a positive definite quadratic form $[A, B, C]$
Output : The reduced form $[A_{\mathrm{fin}}, B_{\mathrm{fin}}, C_{\mathrm{fin}}]$ in $[\mathcal{Q}]$

$(A_0, B_0, C_0) \leftarrow (A, B, C)$
if $C_0 < A_0$ then $(C_0, A_0) \leftarrow (A_0, C_0)$
for $j = 1, \ldots, n$ do
    if $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ then
        $i_j \leftarrow \lfloor \log_2 |B_{j-1}| \rfloor - \lfloor \log_2 A_{j-1} \rfloor - 1$
        $m \leftarrow 2^{i_j}$
        $A_j \leftarrow C_{j-1} - (\mathrm{sgn}(B_{j-1}))mB_{j-1} + m^2 A_{j-1}$,
          $B_j \leftarrow B_{j-1} - (\mathrm{sgn}(B_{j-1}))2mA_{j-1}$, $C_j \leftarrow A_{j-1}$
    end if
    else
        $(A_j, B_j, C_j) \leftarrow (A_{j-1}, B_{j-1}, C_{j-1})$
    end if
    if $C_j < A_j$ then $(C_j, A_j) \leftarrow (A_j, C_j)$
end for
if $A_n < |B_n|$ then
    $A_{n+1} \leftarrow C_n - (\mathrm{sgn}(B_n))B_n + A_n$, $B_{n+1} \leftarrow B_n - 2(\mathrm{sgn}(B_n))A_n$, $C_{n+1} \leftarrow A_n$
end if
else $(A_{n+1}, B_{n+1}, C_{n+1}) \leftarrow (A_n, B_n, C_n)$
if $C_{n+1} < A_{n+1}$ then $(C_{n+1}, A_{n+1}) \leftarrow (A_{n+1}, C_{n+1})$
if $A_{n+1} = -B_{n+1}$ then $(A_{\mathrm{fin}}, B_{\mathrm{fin}}, C_{\mathrm{fin}}) \leftarrow (A_{n+1}, -B_{n+1}, C_{n+1})$
else $(A_{\mathrm{fin}}, B_{\mathrm{fin}}, C_{\mathrm{fin}}) \leftarrow (A_{n+1}, B_{n+1}, C_{n+1})$
return $(A_{n+1}, B_{n+1}, C_{n+1})$
```

5.2. **Quantum Algorithm for Binary Reduction on Quadratic Forms.** Our quantum algorithm for binary reductions on quadratic forms is shown in Algorithm 3, which is based on Algorithm 2. Algorithm 3 uses Algorithm 4 as subroutines. For a predicate P (e.g., $P(A, B) := \neg(|B| \leqslant 2A)$), by $[P]_{\mathsf{bool}}$ we denote the bit that is 1 if the predicate P holds and 0 if P does not hold.

Algorithm 3 is obtained from Algorithm 2 by not only converting every operation into a reversible one but also modifying how to compute (A_j, B_j, C_j) from $(A_{j-1}, B_{j-1}, C_{j-1})$, which we explain below.

Let $(A'_j, B'_j, C'_j) := (C_{j-1} - (\mathrm{sgn}(B_{j-1}))mB_{j-1} + m^2A_{j-1}, B_{j-1} - (\mathrm{sgn}(B_{j-1}))2mA_{j-1}, A_{j-1})$. Algorithm 2 computes (A'_j, B'_j, C'_j) only if $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ holds and sets $(A_j, B_j, C_j) := (A'_j, B'_j, C'_j)$ if $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ holds. If $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ does not hold, then Algorithm 2 just sets $(A_j, B_j, C_j) := (A_{j-1}, B_{j-1}, C_{j-1})$. On the other hand, Algorithm 3 computes (A'_j, B'_j, C'_j) regardless of whether $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ holds, and then sets (A_j, B_j, C_j) as (A'_j, B'_j, C'_j) or $(A_{j-1}, B_{j-1}, C_{j-1})$ depending on whether $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ holds. (In fact the new procedure to compute (A'_j, B'_j, C'_j) is mainly performed in Algorithm 4.)

We modified how to compute (A_j, B_j, C_j) to avoid the conditional branch on the relatively heavy procedures (i.e., line 5-7 of Algorithm 2), which reduces the depth of the quantum circuit to some extent. If we do not modify the computation of (A_j, B_j, C_j), then we would obtain the complexity of depth $O(n^2(\log n)^2)$, width $O(n \log n)$, and size $O(n^2 \log n)$, instead of our complexity of depth $O(n^2 \log n)$, width $O(n^2)$, and size $O(n^2 \log n)$.

Algorithm 3 — Quantum algorithm for binary reduction on quadratic forms

```
Input  : A positive definite quadratic form $[A, B, C]$.
Output : The reduced form $[A_{\text{fin}}, B_{\text{fin}}, C_{\text{fin}}]$ in $[\mathcal{Q}]$

// We show how the state changes in comments.
1  Let $(A_0, B_0, C_0) := (A, B, C)$. // The initial state is $|A_0, B_0, C_0\rangle$.
2  Compute and store $b_0 := [C < A]_{\text{bool}}$. // The current state is $|A_0, B_0, C_0\rangle \otimes |b_0\rangle$
3  if $b_0 = 1$ then Swap $A_0$ and $C_0$.
4  for $j = 1, \dots, n$ do
5      Compute and store $b_{j,1} := [\neg(|B_{j-1}| \leqslant 2A_{j-1})]_{\text{bool}}$, $b_{j,2} := [B_{j-1} < 0]_{\text{bool}}$, and
         $|i_j\rangle := \lfloor \log_2 |B_{j-1}| \rfloor - \lfloor \log_2 A_{j-1} \rfloor - 1$. // The current state is $|A_0, B_0, C_0\rangle |b_0\rangle \otimes$
         $\left(\bigotimes_{1 \leqslant k \leqslant j-1} |A'_k, B'_k, C'_k\rangle |A_k, B_k, C_k\rangle |b_{k,1}, b_{k,2}, b_{k,3}\rangle |i_k\rangle\right) \otimes |b_{j,1}, b_{j,2}\rangle |i_j\rangle$.
6      Run Algorithm 4 on $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle$ to obtain
         $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |A'_j, B'_j, C'_j\rangle |A_j, B_j, C_j\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle$. // The current state is
         $|A_0, B_0, C_0\rangle |b_0\rangle \otimes \left(\bigotimes_{1 \leqslant k \leqslant j-1} |A'_k, B'_k, C'_k\rangle |A_k, B_k, C_k\rangle |b_{k,1}, b_{k,2}, b_{k,3}\rangle |i_k\rangle\right) \otimes$
         $|A'_j, B'_j, C'_j\rangle |A_j, B_j, C_j\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle$.
7      Compute and store $b_{j,3} := [C_j < A_j]_{\text{bool}}$.
8      if $b_{j,3} = 1$ then Swap $A_j$ and $C_j$.
       // The current state is
         $|A_0, B_0, C_0\rangle |b_0\rangle \otimes \left(\bigotimes_{1 \leqslant k \leqslant j} |A'_k, B'_k, C'_k\rangle |A_k, B_k, C_k\rangle |b_{k,1}, b_{k,2}, b_{k,3}\rangle |i_k\rangle\right)$.
9  end for
10 Compute and store $b_{n+1,1} := [A_n < |B_n|]_{\text{bool}}$ and $b_{n+1,2} := [B_n < 0]_{\text{bool}}$. Set
     $i_{n+1} := 0$.
11 Run Algorithm 4 on $|A_n, B_n, C_n\rangle |b_{n+1,1}, b_{n+1,2}\rangle |i_{n+1}\rangle$ to obtain
     $|A_n, B_n, C_n\rangle |A'_{n+1}, B'_{n+1}, C'_{n+1}\rangle |A_{n+1}, B_{n+1}, C_{n+1}\rangle |b_{n+1,1}, b_{n+1,2}\rangle |i_{n+1}\rangle$.
12 Compute and store $b_{n+1,3} := [C_{n+1} < A_{n+1}]_{\text{bool}}$.
13 if $b_{n+1,3} = 1$ then Swap $A_{n+1}$ and $C_{n+1}$.
   // The current state is
     $|A_0, B_0, C_0\rangle |b_0\rangle \otimes \left(\bigotimes_{1 \leqslant k \leqslant n+1} |A'_k, B'_k, C'_k\rangle |A_k, B_k, C_k\rangle |b_{k,1}, b_{k,2}, b_{k,3}\rangle |i_k\rangle\right)$.
14 Compute and store $(A_{\text{fin}}, B_{\text{fin}}, C_{\text{fin}}) = (A_{n+1}, (-1)^{\delta(A_{n+1}+B_{n+1}=0)} B_{n+1}, C_{n+1})$ into
     a new register.
15 Uncompute line 2 - line 13 to obtain $|A_0, B_0, C_0\rangle \otimes |A_{\text{fin}}, B_{\text{fin}}, C_{\text{fin}}\rangle$.
```

Algorithm 4 — Subroutine for core reduction steps

```
Input   : A state $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle$, where $A_{j-1}, B_{j-1}, C_{j-1}$ are
          integers, $b_{j,2} \in \{0,1\}$, and $i_j \in \{0,1\}^{\log_2 n}$.[a]
Output  : The state $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |A'_j, B'_j, C'_j\rangle |A_j, B_j, C_j\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle$.
          Here, $(A'_j, B'_j, C'_j) := (C_{j-1} + (-1)^{b_{j,2}+1} 2^{i_j} B_{j-1} + 2^{2i_j} A_{j-1}, B_{j-1} +$
          $(-1)^{b_{j,2}+1} 2^{i_j+1} A_{j-1}, A_{j-1})$. In addition, $(A_j, B_j, C_j) = (A'_j, B'_j, C'_j)$ if
          $b_{j,1} = 1$ and $(A_j, B_j, C_j) = (A_{j-1}, B_{j-1}, C_{j-1})$ if $b_{j,1} = 0$.
// We denote the output registers to write $A'_j$, $B'_j$, and $C'_j$ by out$_A$, out$_B$, and out$_C$, respectively.
   In addition, we denote the register to write $(A_j, B_j, C_j)$ by out$_{(A,B,C)}$.
1 Write a single copy of $B_{j-1}$ and two copies of $A_{j-1}$ into auxilally registers.
  // The current state is $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes |B_{j-1}\rangle |A_{j-1}\rangle |A_{j-1}\rangle$.
2 Multiply $2^{i_j}$, $2^{2i_j}$, and $2^{i_j+1}$ in $B_{j-1}$, $A_{j-1}$, and $A_{j-1}$ in the auxilally registers.
  // The current state is $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes |2^{i_j} B_{j-1}\rangle |2^{i_j} A_{j-1}\rangle |2^{i_j+1} A_{j-1}\rangle$.
3 Compute and write $2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}$ and $-2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}$ into new
  auxilally registers together with $B_{j-1} + 2^{i_j+1} A_{j-1}$ and $B_{j-1} - 2^{i_j+1} A_{j-1}$
  // The current state is $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes$
     $|2^{i_j} B_{j-1}\rangle |2^{i_j} A_{j-1}\rangle |2^{i_j+1} A_{j-1}\rangle |-2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a1}} |2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a2}}$
     $|B_{j-1} + 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a3}} |B_{j-1} - 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a4}}$.
4 Copy the value in the register a1 or a2 (resp., a3 or a4) to the register out$_{A'}$ (resp.,
  out$_{B'}$) depending on whether $b_{j,2} = 1$.
  // The current state is
     $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |(-1)^{b_{j,2}+1} 2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{out}_{A'}} |B_{j-1}(-1)^{b_{j,2}+1} 2^{i_j+1} A_{j-1}\rangle_{\mathsf{out}_{B'}}$
     $|b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes |2^{i_j} B_{j-1}\rangle |2^{i_j} A_{j-1}\rangle |2^{i_j+1} A_{j-1}\rangle$
     $|-2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a1}} |2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a2}} |B_{j-1} + 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a3}} |B_{j-1} - 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a4}}$.
5 Add $C_{j-1}$ to the out$_{A'}$ register and copy $A_{j-1}$ to the out$_{C'}$ register.
  // The current state is $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |A'_j\rangle_{\mathsf{out}_{A'}} |B'_j\rangle_{\mathsf{out}_{B'}} |C'_j\rangle_{\mathsf{out}_{B'}} |b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes$
     $|2^{i_j} B_{j-1}\rangle |2^{i_j} A_{j-1}\rangle |2^{i_j+1} A_{j-1}\rangle |-2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a1}} |2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a2}}$
     $|B_{j-1} + 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a3}} |B_{j-1} - 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a4}}$.
6 Copy $(A_{j-1}, B_{j-1}, C_{j-1})$ or $(A'_j, B'_j, C'_j)$ into the out$_{(A,B,C)}$ register depending on
  whether $b_{j,1} = 0$ or not.
  // The current state is
     $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |A'_j\rangle_{\mathsf{out}_{A'}} |B'_j\rangle_{\mathsf{out}_{B'}} |C'_j\rangle_{\mathsf{out}_{B'}} |A_j, B_j, C_j\rangle_{\mathsf{out}_{(A,B,C)}} |b_{j,1}, b_{j,2}\rangle |i_j\rangle \otimes$
     $|2^{i_j} B_{j-1}\rangle |2^{i_j} A_{j-1}\rangle |2^{i_j+1} A_{j-1}\rangle |-2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a1}} |2^{i_j} B_{j-1} + 2^{i_j} A_{j-1}\rangle_{\mathsf{a2}}$
     $|B_{j-1} + 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a3}} |B_{j-1} - 2^{i_j+1} A_{j-1}\rangle_{\mathsf{a4}}$.
7 Uncompute lines 1-3.
  // The current state is
     $|A_{j-1}, B_{j-1}, C_{j-1}\rangle |A'_j\rangle_{\mathsf{out}_{A'}} |B'_j\rangle_{\mathsf{out}_{B'}} |C'_j\rangle_{\mathsf{out}_{B'}} |A_j, B_j, C_j\rangle_{\mathsf{out}_{(A,B,C)}} |b_{j,1}, b_{j,2}\rangle |i_j\rangle$.
```

[a]In fact i_j may be negative if $b_{j,1} = 0$, but we regard i_j as an unsigned $\log_2 n$-bit integer.

5.3. **Complexity Analysis for Algorithm 3.** This section provides complexity analysis for Algorithm 3. To be precise, we show the following theorem.

Theorem 5.1. Algorithm 3 can be implemented on a quantum circuit of depth $O(n \log n)$, width $O(n^2)$, and size $O(n^2 \log n)$. More precisely, the required Toffoli depth, ancillary qubits, and Toffoli gates in total are at most $(568n \log n + 896n)$, $(7n^2 + 26n)$, and $(144n^2 \log n + 2834n^2)$.

The theorem shows that our algorithm is asymtptically faster than its classical counterpart, which would be of depth $\mathrm{O}(n^2)$. Another interesting comparison is with the GCD circuit design in [8]. Indeed quadratic form reduction generalize GCD problem and yet we managed to obtain the same depth complexity. First, we analyze complexity of Algorithm 4.

Lemma 5.2. Algorithm 4 can be implemented on a quantum circuit of depth $O(\log n)$, width $O(n)$, and size $O(n \log n)$. More precisely, the required Toffoli depth, ancillary qubits, and Toffoli gates in total are at most $(342 \log n + 53)$, $(8n + n \log n + 6)$, and $(72n \log n + 343n)$, respectively.

Proof. We assume operations in each line are done in a sequential manner.

First, we analyze the depth. Lines 1 do not require any Toffoli gates. Line 2 can be done by applying the rotation algorithm in Section 4.2 three times, which requires $3 \times 12\lceil \log n \rceil = 36\lceil \log n \rceil$ Toffoli depth. Line 3 performs 2 additions and 2 subtractions, which can be done by $4 \times 30 \log n = 120 \log n$ Toffoli depth. By Lemma 2.1, line 4 can be performed by Toffoli depth 4. Line 5 performs a single addition, which requires $30 \log n$ Toffoli depth. By Lemma 2.1 Line 6 can be performed by Toffoli depth 2. In total, Toffoli depth required is at most $2 \times (36\lceil \log n \rceil + 120 \log n) + 4 + 30 \log n + 2 \leqslant 342 \log n + 78$.

Next, we analyze the total number of Toffoli gates. Line 1 does not require any Toffoli gates. Line 2 uses $3 \times 12n\lceil \log n \rceil = 36n\lceil \log n \rceil$ Toffoli gates. Line 3 requires $4 \times 29n = 116n$ Toffoli gates in total. Line 4 uses $4n$ Toffoli gates. Line 5 requires $29n$ Toffoli gates. Line 6 uses $6n$ Toffoli gates. In total, the number of Toffoli gates is at most $2 \times (36n\lceil \log n \rceil + 116n) + 4n + 29n + 6n = 72n\lceil \log n \rceil + 271n \leqslant 72n \log n + 343n$.

Finally, we study how many ancilla quibts are required. The number of ancillary qubits explicitly mentioned in the description of Algorithm 4 is $7n$ in total. To perform additions and subtractions (resp., multiplications (rotations), and copy of data), the additional number of ancillary qubits required is at most $(n + 3n/\log n)$ (resp., $n\lceil \log n \rceil$ and n). Thus the total number of ancillary qubits required is at most $7n + \max\{n + 3n/\log n, n\lceil \log n \rceil, n\} = 7n + (n\lceil \log n \rceil + 6) \leqslant 8n + n \log n + 6$.

Even if we count all the gates (but not only Toffoli gates), by similar arguments we can deduce that the depth is in $O(\log n)$, width is in $O(n \log n)$, and the size is in $O(n \log n)$. □

Proof of Theorem 5.1.
Depth. Here we analyze Toffoli depth, explaining how the computations of each line are performed in detail.

In line 2, we assume $b_0 = [C < A]_{\mathsf{bool}}$ is computed as follows. (i) Compute $C - A$. Then, the most significant bit of $C - A$ matches $[C < A]_{\mathsf{bool}}$. (ii) copy this bit into the output register. (iii) uncompute the subtraction. These operations can be done with Toffoli depth $2 \times 30 \log n = 60 \log n$, $3n/\log n$ ancillary qubits, and $2 \times 29 = 58n$ Toffoli gates in total.
Line 2 of Algorithm 3 can be done with Toffoli depth $60 \log n$.

Recall that bit swap $|b_1\rangle |b_2\rangle \mapsto |b_2\rangle |b_1\rangle$ $(b_1, b_2 \in \{0,1\})$ can be performed by sequentially applying CNOT gates three times. We assume Line 3 is performed by sequentially applying the operation $|b\rangle |x\rangle |y\rangle \mapsto |b\rangle |x\rangle |y \oplus b \cdot x\rangle$ three times in the same way. Then Line 3 requires at most Toffoli depth 3 by Lemma 2.1.

In line 5, we assume the computations are performed as follows:

(i) Compute $b_{j,2}$ by just copying the most significant bit of B_{j-1}.
(ii) Compute $|B_{j-1}|$ into a new auxiliary register and multiply A_{j-1} with 2 (in parallel).
(iii) Subtract $2A_{j-1}$ from $|B_{j-1}|$.
(iv) Compute $b_{j,1}$ by copying the most significant bit of $|B_{j-1}| - 2A_{j-1}$ and then flipping it.
(v) Uncompute (iii).
(vi) Compute and write $\lfloor \log |B_{j-1}| \rfloor$ and $\lfloor \log 2A_{j-1} \rfloor$ into new auxiliary registers in parallel.
(vii) Compute $i_j = \lfloor \log |B_{j-1}| \rfloor - \lfloor \log 2A_{j-1} \rfloor$.
(ix) Uncompute (vi) and (ii).

Then, (i) does not use any Toffoli gates. (ii) can be done with Toffoli depth $(60 \log n + 2)$. (iii) can be performed with $30 \log n$ Toffoli depths. (iv) does not use any Toffoli gates. $8\lceil \log_2 n \rceil$ Toffoli depths is sufficient for (vi). (vii) can be performed with at most $30 \log n$ Toffoli depths. In summary, the computations of line 5 can be performed with Toffoli depth $2 \times (60 \log n + 2) + 2 \times 30 \log n + 2 \times 8\lceil \log_2 n \rceil + 30 \log n \leqslant 226 \log n + 20$.

Line 6 requires at most $(342 \log n + 53)$ Toffoli depth by Lemma 5.2.

Line 7 and line 8 are computed in the same way as line 2 and line3, which can be performed with Toffoli depth $60 \log n$ and 3, respectively.

We assume line 10 is computed as follows: (i) Compute $b_{n+1,2}$ (by just copying the most significant bit of B_n.) (ii) Compute $|B_n|$ and write it into an auxiliary register, then subtract it from A_n. (iii) Compute $b_{n+1,1}$ (by just copying the most significant bit of $|B_n| - A_n$). (iv) Uncompute (ii). These computations require Toffoli depths at most $2 \times (60 \log n + 2 + 30 \log n) = 180 \log n + 4$.

Line 11 requires at most $(342 \log n + 53)$ Toffoli depth by Lemma 5.2. Line 12 and line 13 are computed in the same way as line 2 and line3, which can be performed with Toffoli depth $60 \log n$ and 3, respectively.

We assume line 14 is computed as follows. (i) compute $A_{n+1} + B_{n+1}$ and write it to an auxiliary register. (ii) Take OR of all the bits of $A_{n+1} + B_{n+1}$ to compute $\delta(A_{n+1} + B_{n+1} = 0)$ by using n auxiliary qubits and n OR gates. Here, the OR gate is a three-bit gate computing $|x\rangle |y\rangle |z\rangle \mapsto |x\rangle |y\rangle |z \oplus (x \vee y)\rangle$, which can be implemented with a single Toffli gate and 6 NOT gates[5]. (iii) Compute $-B_{n+1}$ and write it into an auxiliary register. (iv) Copy A_{n+1}, B_{n+1} or $-B_{n+1}$ depending whether $\delta(A_{n+1} + B_{n+1} = 0) = 0$, and C_{n+1}, into the auxiliary register. Uncompute (i)-(iii). These computetions can be performed with Toffoli depth $2 \times (30 \log n + n + 30 \log n) + 2 = 120 \log n + 2n + 2$.

In summary, Algorithm 3 requires Toffoli depth at most $2 \times (60 \log n + 3 + n \times ((226 \log n + 20) + (342 \log n + 53) + 60 \log n + 3) + (180 \log n + 4) + 60 \log n + 3) + (120 \log n + 2n + 2) \leqslant 568 n \log n + 154 n + 720 \log n + 22 \leqslant 568 n \log n + 896 n$.

Size. Next, we count the number of all the Toffoli gates required in total. Line 2 uses at most $2 \times 29 = 58n$ Toffoli gates in total. Line 3 uses at most $3n$ Toffoli gates.

In executing line 5, (i) does not use any Toffoli gates. (ii) uses at most $60n$ Toffoli gates. (iii) uses at most $29n$ Toffoli gates. (iv) does not use any Toffoli gates. (vi) uses $2 \times 8n = 16n$ Toffoli gates in total. (vii) uses at most $29n$ Toffoli gates. In summary, line 5 uses at most $2 \times 60n + 2 \times 29n + 2 \times 16n + 29n = 239n$ Toffoli gates in total.

Line 6 can be performed with at most $(72 n \log n + 343 n)$ Toffoli gates. Line 7 and Line 8 use at most $58n$ and $3n$ Toffoli gates, respectively.

In executing Line 10, (i) does not require any Toffoli gates. (ii) uses at most $60n + 29n = 89n$ Toffoli gates. (iii) does not use any Toffoli gates. In summary, line 10 uses at most $2 \times 89n = 178n$ Toffoli gates.

[5]By applying Toffoli on $|\bar{x}\rangle |\bar{y}\rangle |\bar{z}\rangle$ we obtain $|\bar{x}\rangle |\bar{y}\rangle \overline{|z \oplus (x \oplus y)\rangle}$.

Line 11, line 12, and line 13 use at most $58n$, $3n$, and $(72n\log n + 343n)$ Toffoli gates, respectively.

In executing line 14, (i) uses at most $29n$ Toffoli gates. (ii) uses at most n Toffoli gates. (iii) uses at most $29n$ Toffoli gates. (iv) can be performed with at most $2n$ Toffoli gates. In summary, line 10 uses at most $2 \times (29n + n + 29n) + 2n = 120n$ Toffoli gates in total.

As a result, the number of Toffoli gates required is at most $2 \times (58n + 3n + n \times (239n + (72n\log n + 343n) + 58n + 3n) + 178n + 58n + 3n + (72n\log n + 343n)) + 120n = 144n^2\log n + 1286n^2 + 142n\log n + 1406n \leqslant 144n^2\log n + 2834n^2$.

ancillary qubits. The number of ancillary qubits explicitly mentioned in (the comments of) Algorithm 3 is at most $(n+1) \times (7n+3)$. In addition, it is straightforward to check that additional $(n\log n + 20n)$ ancilla quibts are sufficient to compute all the intermediate computations. Hence the total number of ancilla quibts required is at most $(n+1) \times (7n+3) + (n\log n + 20n) = 7n^2 + 23n + 3 \leqslant 7n^2 + 26n$.

It is easy to check that the asymptotic complexity does not change even if we count not only Toffoli gates but also all other gates. □

6. Application to dimension 2 Lattice Reduction

6.1. **Two dimensional lattices.**

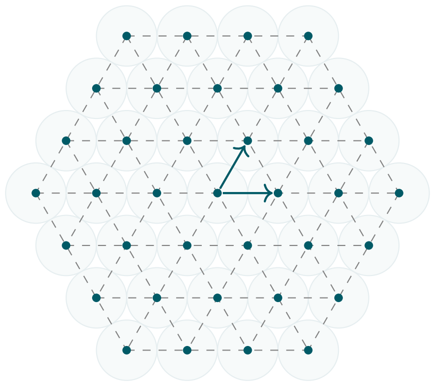


FIGURE 12. Example of a two dimensional lattice (the hexagonal lattice)

6.1.1. *Lattices.* Given two linearly independent vectors (u, v) of a Euclidean space $(V, \|\cdot\|)$, the two dimensional lattice spanned by u and v is the subgroup $\mathcal{L}$ of V spanned by these two elements, that is to say $\mathcal{L} = u\mathbb{Z} \oplus v\mathbb{Z}$. Topologically, this group is discrete in V for the norm $\|\cdot\|$, implying that there exists a finite number of vectors with minimal norm in $\mathcal{L}$. Such vectors are called the shortest vectors of the lattice $\mathcal{L}$. Given an arbitrary basis of $\mathcal{L}$, a classical problem consists in retrieving the shortest vectors.

6.2. **Reduction process.**

This task, called lattice reduction, can be performed efficiently by an iterative process called Gauss-Lagrange reduction.

Algorithm 5 — Gauss reduction

```
Input  : $(u, v)$ a basis of a two-dimensional lattice $\mathcal{L}$
Output : A reduced basis $(u, v)$ of $\mathcal{L}$

1 if $\|v\| < \|u\|$ then return Gauss$(v, u)$
2 while Progress is done do
3     $v' \leftarrow v - \left\lfloor \frac{\langle u,v\rangle}{\|u\|^2} \right\rceil u$
4     $u, v \leftarrow v', u$
5 end while
6 return $(u, v)$
```

Without loss of generality, we can suppose that $\|u\| \leqslant \|v\|$, by rearranging the order of these vectors. Now that the first vector of the basis is the smallest among the two, we try to reduce the norm of the second one. Explicitly we want to find the shortest vector v' such that (u, v') is a basis of $\mathcal{L}$. The problem is now reduced to finding the shortest vector in the coset $v + u\mathbb{Z}$. A straightforward computation ensures that this element is $v - \left\lfloor \frac{\langle u,v\rangle}{\|u\|^2} \right\rceil u$. We can then restart this procedure until no reduction of the size of the vectors can be done. The corresponding pseudo-code is given in Algorithm 5. Remark that the step operation of this reduction consists in applying the linear transformation

$$S = \begin{pmatrix} 0 & 1 \\ 1 & t(\mathcal{Q}) \end{pmatrix}, \qquad \text{where } \mathrm{t}(\mathcal{Q}) = -\left\lfloor \frac{\langle u,v\rangle}{\|u\|^2} \right\rceil,$$

to the current basis.

Let (u, v) be a basis of a lattice $\mathcal{L}$ output by the Gauss algorithm. By construction, these vectors satisfy

$$\|u\| \leqslant \|v\|$$

and:

$$\left\lfloor \frac{\langle u,v\rangle}{\|u\|^2} \right\rceil = 0, \qquad \text{i.e} \qquad |\langle u,v\rangle| \leqslant \frac{\|u\|^2}{2}.$$

From these observations, we can give an axiomatic definition of a reduced basis:

Definition 6.1 (Gauss-reduced basis)**.** Let $\mathcal{L}$ be a two dimensional real lattice. A basis (u, v) of $\mathcal{L}$ is said to be *Gauss-reduced* if it fulfills the two conditions:

(1) $\|u\| \leqslant \|v\|$
(2) $|\langle u,v\rangle| \leqslant \frac{\|u\|^2}{2}$.

Proposition 6.2. Let (u, v) be a reduced basis of a lattice $\mathcal{L}$, then u is a shortest vector of $\mathcal{L}$.

Theorem 6.3. The number of steps of the Gauss-reduction is $\mathrm{O}(M)$, where M is a bound on the number of bits required to represents the basis given as input.

The proof of these theorems is given in appendix for completeness.

6.3. **Relation between lattices and quadratic forms.** The close resemblance between the operations used to reduced quadratic forms and plane lattices is not a coincidence: lattices and (positive definite) forms are avatar of the same objects. More precisely, let (u, v) be a basis of a plane lattice $\mathcal{L}$, and $\mathcal{Q}$ the form encoded by the Gram matrix of (u, v):

$$Q = \begin{pmatrix} \langle u, u\rangle & \langle u, v\rangle \\ \langle u, u\rangle & \langle v, v\rangle \end{pmatrix}.$$

Then we have the following correspondence:

	Lattice formalism $\mathcal{L}$	Quadratic form formalism
Object	Basis $M = (u, v)$	Gram matrix $G = M^t M$
Step operation	$M \leftarrow MS_\lambda$	$G \leftarrow S_\lambda^t G S_\lambda$
Reduceness condition	$\|\|v\|\|^2 \geqslant \|\|u\|\|^2 \geqslant 2\|\langle u, v\rangle\|$	$C \geqslant A \geqslant \|B\|$

A quick look at the reduceness condition reveals that M is reduce in the lattice sense is equivalent for $Q = M^t \cdot M$ to be reduced as a Quadratic form. The Cauchy-Schwarz inequality ensures that $M^t \cdot M$ is a positive definite form. Reciproqually, given a positive definite form represented by a matrix Q, the so-called Cholesky decomposition of Q ensures an upper triangular matrix R such that $Q = R^T R$. Th columns of R are a basis of the lattice $Q\mathbb{Z}^2$, which corresponding Gram-matrix is Q itself. Henceforth, reducing plane lattices or positive definite quadratic forms is equivalent.

Due to the natural correspondence between two-dimensional lattices and quadratic forms with $\Delta \leqslant 0$, the algorithms for quadratic forms in the previous section can easily be converted into those for two-dimensional lattices. Recall that a basis (u, v) of a lattice corresponds to the quadratic form $[\|u\|^2, 2\langle u, v\rangle, \|v\|^2]$. Algorithm 6 shows a lattice version of Algorithm 2. As internal states, Algorithm 6 maintains the information of vectors u_j, v_j, together with their norms $\|u_j\|^2, \|v_j\|^2$ and the inner product $\langle u_j, v_j\rangle$ (which correspond to A_j, C_j, and B_j in Algorithm 2, respectively). Note that Algorithm 6 does not perform division but require several multiplications to compute the norms and the inner product of the vectors (in line 1).

Algorithm 6 — Binary reduction for two-dimensional lattices

```
Input  : A basis $(u, v)$ of a two-dimensional lattice $\mathcal{L} \subseteq \mathbb{Z}^2$
Output : A reduced basis $(\tilde{u}, \tilde{v})$ of $\mathcal{L}$

1  $u_0, v_0, A_0, B_0, C_0 \leftarrow u, v, \|u\|^2, 2\langle u, v\rangle, \|v\|^2$
2  if $C_0 < A_0$ then $(v_0, u_0, C_0, A_0) \leftarrow (u_0, v_0, A_0, C_0)$
3  for $j = 1, \ldots, n$ do
4      if $\neg(|B_{j-1}| \leqslant 2A_{j-1})$ then
5          $i_j \leftarrow \lfloor \log_2 |B_{j-1}| \rfloor - \lfloor \log_2 A_{j-1} \rfloor - 1$  $m \leftarrow 2^{i_j}$
6          $A_j \leftarrow C_{j-1} - (\mathrm{sgn}(B_{j-1}))mB_{j-1} + m^2 A_{j-1}$,
            $B_j \leftarrow B_{j-1} - (\mathrm{sgn}(B_{j-1}))2mA_{j-1}$, $C_j \leftarrow A_{j-1}$
7          $(u_j, v_j) \leftarrow (v_{j-1} - \mathrm{sgn}(B_{j-1})mu_{j-1}, u_{j-1})$
8      end if
9      else
10         $(A_j, B_j, C_j) \leftarrow (A_{j-1}, B_{j-1}, C_{j-1})$, $(u_j, v_j) \leftarrow (u_{j-1}, v_{j-1})$
11     end if
12     if $C_j < A_j$ then $(C_j, A_j) \leftarrow (A_j, C_j)$, $(u_j, v_j) \leftarrow (u_{j-1}, v_{j-1})$
13 end for
14 if $A_n < |B_n|$ then
15     $A_{n+1} \leftarrow C_n - (\mathrm{sgn}(B_n))B_n + A_n$, $B_{n+1} \leftarrow B_n - 2(\mathrm{sgn}(B_n))A_n$, $C_{n+1} \leftarrow A_n$,
16     $(u_{n+1}, v_{n+1}) \leftarrow (v_n - \mathrm{sgn}(B_n)u_n, u_n)$
17 end if
18 else $(A_{n+1}, B_{n+1}, C_{n+1}) \leftarrow (A_n, B_n, C_n)$, $(u_{n+1}, v_{n+1}) \leftarrow (u_n, v_n)$
19 if $C_{n+1} < A_{n+1}$ then $(C_{n+1}, A_{n+1}) \leftarrow (A_{n+1}, C_{n+1})$, $(v_{n+1}, u_{n+1}) \leftarrow (u_{n+1}, v_{n+1})$
20 if $A_{n+1} = -B_{n+1}$ then $(\tilde{u}, \tilde{v}) \leftarrow (u_{n+1}, u_{n+1} + v_{n+1})$
21 else $(\tilde{u}, \tilde{v}) \leftarrow (u_{n+1}, v_{n+1})$
22 return $(\tilde{u}, \tilde{v})$
```

Algorithm 6 can be converted into a quantum circuit the same way as Algorithm 2 is converted into Algorithm 3. In fact, there is a difference between Algorithm 6 and Algorithm 2 in that we have to perform some multiplications at the beginning of Algorithm 6 to compute $\|u\|^2, \langle u, v\rangle, \|v\|^2$ from u and v. Still, the multiplications can be performed on a quantum circuit of depth $O(n \log n)$, width $O(n)$, and size $O(n^2)$, by implementing the ordinary long multiplication of binary numbers with the circuit for addition of depth $O(\log n)$, width $O(n)$, and size $O(n)$. Thus we obtain the following theorem.

Theorem 6.4. Algorithm 6 can be implemented on a quantum circuit of depth $O(n \log n)$, width $O(n^2)$, and size $O(n^2 \log n)$.

7. Conclusion

In the end, we design the first efficient quantum circuit solving definite quadratic form reduction in almost linear width and depth and almost quadratic volume. Due to its application to quantum LLL, together with works form [18], this work will help setting security margin for the postquantum primitives in the lattice based cryptography.

Future Work. Our work only focus on the reduction of definite quadratic form so it is natural to wonder about the indefinite case. Our research seems to indicate that reducing indefinite quadratic forms is harder than definite one. Therefore we are left with the following question : Can we find a reversible shallow circuit that reduces indefinite quadratic forms on a circuit of depth $O(n \log n)$, width $O(n^2)$, and size $O(n^2 \log n)$?

Acknowledgements

We thank Alestair Kay for the LaTeX package quantikz [19], which we used to draw quantum circuits in Section 2, Figures 1-4, 7, 8. In addition, we thank Yasuhiro Takahashi and Seiichiro Tani for helpful comments on quantum circuits for multiplications.

References

[1] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[2] D Mermin. Breaking rsa encryption with a quantum computer: Shor's factoring algorithm. *Lecture notes on Quantum computation*, pages 481–681, 2006.

[3] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[4] Charles H Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *arXiv preprint arXiv:2003.06557*, 2020.

[5] Yasuhiro Takahashi and Noboru Kunihiro. A fast quantum circuit for addition with few qubits. *Quantum Information & Computation*, 8(6):636–649, 2008.

[6] Luis Antonio Brasil Kowada, Renato Portugal, and Celina M. H. de Figueiredo. Reversible karatsuba's algorithm. *J. Univers. Comput. Sci.*, 12(5):499–511, 2006.

[7] Himanshu Thapliyal, Edgard Muñoz-Coreas, T. S. S. Varun, and Travis S. Humble. Quantum circuit designs of integer division optimizing t-count and t-depth. *IEEE Trans. Emerg. Top. Comput.*, 9(2):1045–1056, 2021.

[8] Mehdi Saeedi and Igor L Markov. Quantum Circuits for GCD Computation with $\mathrm{O}(n/logn)$ Depth and $\mathrm{O}(n)$ Ancillae. *arXiv preprint arXiv:1304.7516*, 2013.

[9] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Inf. Comput.*, 6(4):351–369, 2006.

[10] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *Quantum Inf. Comput.*, 10(9&10):872–890, 2010.

[11] Arnold Scönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.

[12] David Harvey and Joris Van Der Hoeven. Integer multiplication in time o (n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

[13] Josef Stein. Computational problems associated with racah algebra. *Journal of Computational Physics*, 1(3):397–405, 1967.

[14] Srijit Dutta, Debjyoti Bhattacharjee, and Anupam Chattopadhyay. Quantum circuits for toom-cook multiplication. *Physical Review A*, 98(1):012311, 2018.

[15] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical algorithm*. Addison Wesley Longman Publishing Co., Inc., 1997.

[16] Ali Akhavi and Brigitte Vallée. Average bit-complexity of euclidean algorithms. *Proceedings of the 27th International Colloquium onAutomata, Languages and Programming, ICALP 2000*, pages 373–387, 07 2000.

[17] Cristopher Moore and Martin Nilsson. Parallel quantum computation and quantum codes. *SIAM J. Comput.*, 31(3):799–815, 2001.

[18] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. *CoRR*, abs/2105.05608, 2021.

[19] Alastair Kay. Tutorial on the quantikz package. *arXiv preprint arXiv:1809.03842*, 2018.

Appendix A. Omitted proofs

Proof of Theorem 6.2. Let (u, v) be a reduced basis of a lattice $\mathcal{L}$. Remark that by definition of the reduction, we have that $\|v\| \leqslant \|v \pm u\|$ and by the analysis of **??**, we have more generally $\|v\| \leqslant \|v + \lambda u\|$ for any integer $\lambda \neq 0$.

Let now $x = \alpha u + \beta v$ be a generic point of $\mathcal{L}$. Clearly if $\beta = 0$ we have $\|x\| \geqslant \|u\|$. We can now suppose without of generality that $\beta > 0$ and set $\alpha = \kappa\beta + \rho$ with $0 \leqslant \rho < \beta$ to be the Euclidean division of α by β. Then we have by reverse triangular inequality:

$$\begin{aligned}\|\alpha u + \beta v\| &\geqslant \beta\|v + \kappa u\| - \rho\|u\| \\ &= (\beta - \rho)\|v + \kappa u\| + \rho(\|v + \kappa u\| - \|u\|) \\ &\geqslant \|v + \kappa u\| \geqslant \|v\| \geqslant \|u\|,\end{aligned}$$

as $\|v + \kappa u\| - \|u\| \geqslant 0$ and $\beta - \rho$ is an integer greater than 0. This ends the proof. □

Proof of Theorem 6.3. Remark first that except for the last iteration of the algorithm, the norm of the first vector of the basis shrinks by a factor of at least $\sqrt{3}^{-1}$. Indeed let (u, v) be the current state of the basis at any step except the final one and (u, v') the basis one step after. Remark that:

$$|\langle u', v'\rangle| = |\langle v, u'\rangle| = \left|\left\lfloor \frac{\langle u, v\rangle}{\|u\|^2} \right\rceil - \frac{\langle u, v\rangle}{\|u\|^2}\right| \|u\|^2 \leqslant \frac{1}{2}\|u\|^2.$$

Suppose that $\|u'\|^2 \geqslant \|u\|/3$, then we would have:

$$|\langle u', v'\rangle| \leqslant \frac{3}{2}\|u'\|^2.$$

In this case, during the next iteration of the algorithm, line 6 of Algorithm 5 makes appear $u' \pm v'$. If this vector appears to be smaller than u' then we would have already computed it differently in the current iteration. Hence, the next step is the final iteration. We can conclude since the norm of the first vector is initially bounded by 2^B. □