Prime Factorization Using Partially Constrained Multiple Quantum Annealing With Analytical and Pattern-Based Variable Reduction

Xinyi Guo, Geguang Miao, Shinichi Nishizawa, Shinji Kimura, *Life Member, IEEE*, and Takashi Sato, *Senior Member, IEEE*

***Abstract*—Factorization of large semiprimes remains one of the most challenging problems for classical computers. Shor's algorithm offers a quantum approach that reduces computational complexity, but its practical application is currently limited by hardware constraints. Meanwhile, as a provisional approach, quantum annealing (QA) has been explored through formulations of the quadratic unconstrained binary optimization (QUBO) problem. Among existing methods, the blockwise partial-product approach effectively reduced the QUBO variable count but was limited to semiprimes up to 21 bits. To extend factorization to larger semiprimes, this paper addresses key engineering challenges in constructing efficient QUBO formulations for prime factorization. We propose five techniques to reduce variable counts and improve scalability with current QA hardware: (1) dividing the problem into subproblems with partial constraints; (2) applying analytical reductions near the LSB; (3) applying analytical reductions near the MSB; (4) exploiting special patterns in semiprimes, with odd bit widths and long MSB-side zero sequences; and (5) balancing variable usage across both sides of the subproblem. Integrated into a QUBO converter, these methods enable stable factorization of semiprimes up to 47-bits within 20 seconds and can extend to special 2049-bit instances with 1001 consecutive MSB-side zeros.**

***Index Terms*—Quantum annealing, QUBO, prime factorization, combinatorial optimization problem, multiplication table, addition of partial products.**

Received 15 May 2024; revised 9 November 2025; accepted 11 January 2026. Date of publication 15 January 2026; date of current version 13 March 2026. This work was supported in part by the "Photonics and Quantum Technology for Society 5.0" by the Cross-ministerial Strategic Innovation Promotion Program (SIP) under the supervision of the National Institute for Quantum Science and Technology (QST), based on the results obtained from a project JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and in part by NEC and JSPS Kakenhi under Grant 23K28052. Recommended for acceptance by K. Gaj. *(Corresponding author: Takashi Sato.)*

Xinyi Guo and Takashi Sato are with the Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan (e-mail: guo.xinyi.68s@st.kyoto-u.ac.jp; takashi@i.kyoto-u.ac.jp).

Geguang Miao and Shinji Kimura are with the Graduate School of Information, Production and Systems, Waseda University, Fukuoka 808-0135, Japan (e-mail: geguangmiao@fuji.waseda.jp; shinji_kimura@waseda.jp).

Shinichi Nishizawa is with the Graduate School of Advanced Science and Engineering, Hiroshima University, Hiroshima 739-0046, Japan (e-mail: nishizawa@hiroshima-u.ac.jp).

Digital Object Identifier 10.1109/TC.2026.3654172

I. Introduction

PRIME factorization (PF) is to find the prime factors p and q for a given semiprime N that satisfy $N = p \times q$. This problem has attracted considerable interest because it is related to public key cryptography [1], wherein messages are encrypted with a public key, which is a semiprime, and can only be decrypted using its private key, which is a pair of prime factors of the semiprime [2]. Despite the existence of various classical algorithms, such as the general number field sieve (GNFS) method [3] recognized as the fastest to date, no deterministic algorithm has yet been developed that solves PF in polynomial time on classical computers.

With the emergence of quantum computing, researchers have explored its potential to tackle PF more efficiently [4]. Shor's algorithm, the most prominent quantum approach, achieves exponential speedup over GNFS by leveraging entanglement and the quantum Fourier transform to factor an integer N in polynomial time [5], [6], [7]. Experimental implementations include factoring $N = 15$ with superconducting qubits [8] and $N = 21$ using qubit recycling [6], as well as optimized order-finding circuits on IBM quantum devices [9]. However, current noisy intermediate-scale quantum (NISQ) hardware suffers from limited number of qubits, short coherence time, and high error rates, which restrict circuit depth and can factor small numbers.

Another model of quantum computing is adiabatic quantum computing (AQC) [10], [11], which has been proven to be polynomially equivalent to the standard gate-based quantum computing model [12]. AQC is also compatible with NISQ devices and offers resilience to certain noise types. As a result, it has been explored for PF in the NISQ era. Using nuclear magnetic resonance (NMR) systems, Peng et al. [13] factored 21, Xu et al. [14] factored 143, and Pal et al. [15] factored 551. However, AQC faces challenges of limited scalability, rapid signal decay with increasing qubit count, and fundamental physical constraints, hindering its application to a large-scale [10].

In an attempt to realize practical quantum computation in the NISQ era, a framework called quantum annealing (QA) has been explored [16]. QA is tailored for combinatorial optimization problems, typically formulated as the minimization of a quadratic unconstrained binary optimization (QUBO) cost function. This feature has motivated several studies on PF using

QA: Jiang et al. [17] used a blockwise binary multiplication table to reduce carries and successfully factored 376,289 (19-bit) using 94 variables. Peng et al. [18] computed the maximum carry and optimized local fields and couplings, factoring 1,005,973 (20-bit) with only 89 variables using D-Wave and the qbsolv package. Wang et al. [19] further extended this approach, factoring 1,630,729 (21-bit) with 125 variables by analyzing different column arrangements in multiplication tables. Although these methods do not reduce the underlying computational complexity, their heuristic strategies have substantially increased the size of solvable semiprimes.

This work investigates how far QA-based factoring can be extended by proposing five methods that reduce variable counts and problem size, thereby improving the tractability of large semiprimes. The first method divides the problem into subproblems with partial constraints, which are solved independently by multiple annealing processes. The second and third methods apply analytical reductions near the LSB and MSB, respectively. The fourth method exploits structural patterns in semiprimes, with odd bit widths and long MSB-side zero sequences, to further compress variables. The fifth method balances variable usage across both sides of each subproblem to reduce overall problem complexity. Notably, the fourth and fifth methods introduce novel pattern-based variable reduction and balancing strategies that have not been addressed in previous work [20]. Implemented in a QUBO converter for Amplify AE [21], these methods enable stable factorization of 47-bit semiprime instances within 20 seconds and extend to special 2049-bit instances with 1001 consecutive MSB-side zeros, highlighting how engineering optimizations can fully leverage the capabilities of current QA hardware. In addition, we analyze the influence of penalty values, annealing time, subproblem configuration, and coefficient tuning. These findings not only reveal the practical limits of QA-based factorization on existing NISQ devices but also provide insights for future algorithm-hardware co-design as QA technology continues to evolve.

II. Preliminary

A. Adiabatic Quantum Computing and Quantum Annealing

AQC solves problems by exploiting the adiabatic theorem of quantum mechanics, which states that a quantum system remains in its ground state if its Hamiltonian evolves sufficiently slowly. In this idealized framework, computation proceeds through a continuous and perfectly adiabatic evolution of a closed quantum system, ensuring that the final ground state encodes the solution of the problem [10].

QA, on the other hand, is a practical realization inspired by the same principle but adapted to hardware limitations. Unlike AQC, QA does not necessarily satisfy the strict conditions of the adiabatic theorem. Current QA implementations—such as D-Wave's quantum annealers [16]—typically operate with fixed annealing schedules that may not be slow enough to maintain adiabaticity for large-scale problems and run at finite temperatures. Consequently, QA should be regarded as a heuristic, non-adiabatic approximation of the adiabatic process rather than its exact physical realization.

B. Quantum Annealer

QA devices fall into two categories: superconducting quantum annealers (e.g., D-Wave [16]) and quantum-inspired annealing machines (e.g., Fixstars' Amplify AE [21]). Quantum annealers use a heuristic cost function-minimization process that searches for low-energy configurations representing problem solutions. D-Wave's hardware has evolved from the Chimera topology (6-way connectivity) to Pegasus (15-way) and Zephyr (20-way), enhancing connectivity, coherence, and solution speed across successive generations [16], [22]. Amplify AE, a GPU-based annealing engine, can handle fully connected QUBO problems exceeding 100,000 binary variables. Although implemented as a simulated QA system, it enables rapid prototyping and scalable optimization tasks on classical hardware.

C. Quadratic Unconstrained Binary Optimization

In practical QA applications, problems are formulated as quadratic unconstrained binary optimization (QUBO) problems, which are mathematically equivalent to Ising models. The objective function is expressed as $f = \sum_i Q_{i,i}x_i + \sum_{i<j} Q_{i,j}x_ix_j$, where $x_i, x_j \in \{0, 1\}$ denote binary variables. $Q_{i,i}$ represents the linear coefficients, and $Q_{i,j}$ denotes the quadratic coefficients. The QA solver determines values of x_i and x_j that minimize f. When mapping optimization problems into QUBO form, higher-order terms beyond quadratic may arise. These terms are converted to quadratic form through the introduction of auxiliary variables and penalty functions [23].

A commonly used penalty-based transformation is:

$$x_1x_2x_3 = yx_3 + P \cdot (x_1x_2 - 2x_1y - 2x_2y + 3y),$$

where the penalty term $P \cdot (x_1x_2 - 2x_1y - 2x_2y + 3y)$ evaluates to zero when $y = x_1x_2$ and introduces a positive penalty otherwise, thereby enforcing the constraint indirectly. This transformation introduces auxiliary variables, such as y, to preserve the quadratic form of the objective function. The penalty constant P controls the strength of constraint enforcement.

Selecting an appropriate value for P is particularly challenging. If P is too large, the penalty term may dominate the objective function and distort the energy landscape, hindering effective exploration and trapping the solver in local minima. Conversely, if P is too small, the penalty may fail to sufficiently enforce the constraint, leading to invalid or infeasible solutions. Determining a proper penalty value is therefore non-trivial and often problem-dependent [23].

III. Related Work

A. Modified Multiplication Table

Jiang et al. [17] transformed the PF problem into a combinatorial optimization task by introducing a blockwise binary multiplication table. In this approach, multiple columns are grouped into blocks to reduce the number of carry variables. Each block's sum—including the relevant product terms ($p \times q$) and carries—is constrained to match the sum of the corresponding bits of the semiprime N in that block. Since N is odd, the

TABLE I
MODIFIED MULTIPLICATION TABLE FOR FACTORING 143

Column	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Block		III			II	I		
p					1	p_2	p_1	1
q					1	q_2	q_1	1
$p \times q$					1	p_2	p_1	1
				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
Carries		c_4	c_3	c_2	c_1			
N	1	0	0	0	1	1	1	1

TABLE II
MULTIPLICATION TABLE DIVIDED INTO THREE BLOCKS FOR 143

Column	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Block		III			II		I	
p					1	p_2	p_1	1
q					1	q_2	q_1	1
$p \times q$					1	p_2	p_1	1
				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
Carries		c_3	c_2			c_1		
N	1	0	0	0	1	1	1	1

least significant bits (LSBs) of the two factors are fixed to 1. The two factors are assumed to have equal bit widths, as similar sizes increase computational complexity, whereas a large bit-width imbalance may weaken cryptographic security. Taking $N = 143$ as an example, Table I illustrates how combining two columns into a single block. The blocks, labeled I-III from LSB to MSB, exclude the 2^0 column, which contains no variables. The corresponding equations are derived as follows:

$$f_1 : p_1 + q_1 + 2p_2 + 2p_1q_1 + 2q_2 - (8c_2 + 4c_1) = (11)_2$$
$$f_2 : (1)_2 + p_2q_1 + p_1q_2 + (1)_2 + 2q_1 + 2p_2q_2 + 2p_1 + c_1 + 2c_2 - (8c_4 + 4c_3) = (01)_2$$
$$f_3 : q_2 + p_2 + 2 \times (1)_2 + c_3 + 2c_4 = (100)_2$$

To solve the aforementioned equations, each equality constraint $x = y$ is transformed into a minimization problem by converting it to $(x - y)^2$. These squared terms are then aggregated into a single objective function. The goal is to find variable assignments that minimize the total sum, thereby driving each squared term to zero. The resulting objective function f is defined as:

$$f = \sum_{i=1}^{3} (\text{Left Hand Side of } f_i - \text{Right Hand Side of } f_i)^2.$$

B. Modified Multiplication Table With Optimized Parameters

Peng et al. [18] proposed a *modified multiplication table* with optimized parameters, based on the same structural format as Table I. In this method, the maximum carry value for each block, denoted as $MaxCarry_i$, is estimated by assuming all partial products and incoming carries are 1, thereby calculating the maximum possible sum for the i-th block. Given the known value of the semiprime in each block, this estimation allows certain carry variables to be set to 0 when the block's maximum possible sum exceeds the semiprime's actual value. For example, in Block I of Table I, the semiprime contributes a sum of 6, while $MaxCarry_1$ is 12. Since $6 = (12 - 6) = (110)_2$, and the binary value $(110)_2$ reflects the number of bits needed to represent the block's value, it follows that both c_1 and c_2 can be safely set to 0. This significantly reduces the number of carry variables required in the QUBO formulation.

IV. PROPOSED METHODS

Jiang et al. [17] and Peng et al. [18] employed the *modified multiplication table* method described in Section III, combining all block equations into a single QUBO objective function for annealing. However, the size of solvable semiprimes was limited, as the D-Wave 2000Q could only handle inputs up to 20 bits. To overcome this limitation and enable larger instances, we propose five methods:

(A) multiple annealing for partial block (MulBlocks),
(B) applying analytical variable reduction for LSB (VarLSB),
(C) applying analytical variable reduction for MSB (VarMSB),
(D) exploitation of special patterns with odd bit widths and long MSB-side zero sequences (SpePattern),
(E) optimal block balancing which balances variable usage across both sides of each subproblem (OBB).

A. Multiple Annealing for Partial Blocks (MulBlocks)

The MulBlocks strategy factors a semiprime by dividing the modified multiplication table into blocks and solving them separately. Instead of performing a single annealing run on the entire problem, MulBlocks follows a two-step process: each block is first solved independently through annealing, and the results are combined to obtain the final factors.

As shown in Table II, the multiplication table is divided into three blocks (I, II, and III), each converted into a QUBO and annealed independently. Block I yields (p_1, q_1), Block II yields (p_1, p_2, q_1, q_2), and Block III yields (p_2, q_2). Block II involves the most complex objective function due to many quadratic terms and auxiliary variables, whereas Blocks I and III are simpler and more compact. To reduce complexity, only Blocks I and III are used in our implementation.

Taking $N = 143$ as an example, the table is split at bit positions 1, 2, and 5, corresponding to columns 2^1, 2^2, and 2^5. The factors p and q are represented as $p = (1p_n \dots p_1 1)_2$ and $q = (1q_n \dots q_1 1)_2$, with equal bit widths. The maximum carry value in each block is determined by assigning 1 to all relevant variables, including carries from previous blocks.

To balance the number of variables across blocks, we allocate $n - \lfloor \frac{n}{2} \rfloor$ bits of p and q to Block I, and $\lfloor \frac{n}{2} \rfloor$ bits to Block III. For example, Block I contains (p_1, q_1) and Block III contains

(p_2, q_2) in the 143-bit case. The resulting objective functions are:

$$f_1 = (p_1 + q_1 - 2c_1 - (1)_2)^2$$
$$f_3 = (2 \times (1)_2 + p_2 + q_2 + c_2 + 2c_3 - (100)_2)^2$$

The QUBOs for f_1 and f_3 are solved independently on the annealing machine, generating candidate solutions. These are then combined and validated against the target semiprime. We adopt a simple exhaustive combination as the baseline, though this step can be accelerated using classical post-processing, such as GCD filtering or heuristic pruning.

As the semiprime bit width increases, the number of variables in Blocks I and III also grows, leading to increased complexity in the QUBO formulation. To further reduce the number of variables and simplify the annealing process, we propose subdividing large blocks into smaller pieces. The resulting objective functions for Pieces 1 and 2 are given as follows:

$$f_1 = \sigma_1 \times f_{11} + \sigma_2 \times f_{12}$$

where f_{11} and f_{12} are the squared constraint equations for each piece, and σ_1, σ_2 are tunable coefficients. Selecting appropriate σ values is critical for ensuring effective QUBO performance and is evaluated in Section V-E.

Essentially, MulBlocks represents a hybrid approach that combines QA with classical computation. QA is employed to filter the solution space by identifying candidates that satisfy partial constraints from Blocks I and III, thereby significantly reducing the search space (candidate solutions). The final factorization is then completed by a classical processor, which efficiently searches through the reduced set of candidates to identify the correct prime factors.

B. Analytical Variable Reduction for LSB (VarLSB)

To further reduce the number of variables in QUBO, we exploit the patterns inherent to the multiplication table. Because no carry-in exists in the LSB part, the sum of product terms of variables is directly related to the pattern of their corresponding bit position of semiprime. We refer to this method as analytical variable reduction for LSB (VarLSB).

Table III presents an example where columns 2^2 and 2^1 of the semiprime are $(00)_2$. From column 2^1, the equation $p_1 + q_1 = 0 + 2c_{12}$ implies that: if $c_{12} = 0$, then $p_1 = q_1 = 0$; if $c_{12} = 1$, then $p_1 = q_1 = 1$. directly yielding $p_1 = q_1$. Substituting q_1 by p_1 into column 2^2, the expression becomes $p_2 + 2p_1 + q_2 = 2c_{23} + 4c_{24}$, leading similarly to $p_2 = q_2$.

A comprehensive analysis of possible bit patterns in columns 2^2 and 2^1 reveals the following four cases that allow further reduction:

$(00)_2$: $p_1 = q_1$; $p_2 = q_2$	$(01)_2$: $p_1 + q_1 = 1$; $p_2 = q_2$
$(10)_2$: $p_1 = q_1$; $p_2 + q_2 = 1$	$(11)_2$: $p_1 + q_1 = 1$; $p_2 + q_2 = 1$

By including the column 2^3, the following four cases out of total eight cases provide the relations of variables:

$(001)_2$, $(010)_2$: $p_3 = q_3$	$(101)_2$, $(110)_2$: $p_3 + q_3 = 1$; $p_3 q_3 = 0$

These relationships reduce both the number of variables and the degree of QUBO terms. For example, a cubic term like $p_3 p_1 q_1$ typically requires an auxiliary variable. However, with $p_1 = q_1$ derived from VarLSB, it simplifies to the quadratic term $p_3 p_1$, avoiding the need for extra variables.

TABLE III
MULTIPLICATION TABLE FOR LSB THE COLUMN

Column	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
p						1	p_3	p_2	p_1	1
q						1	q_3	q_2	q_1	1
$p \times q$					q_1	1	p_3	p_2	p_1	1
				q_2	$p_3 q_2$	$p_3 q_1$	$p_2 q_1$	$p_1 q_1$	q_1	
			q_3	$p_3 q_3$	$p_2 q_3$	$p_2 q_2$	$p_1 q_2$	q_2		
		1	p_3	p_2	p_1	$p_1 q_3$	q_3			
						1				
Carries	c_{89}	c_{78}	c_{67}	c_{56}	c_{45}	c_{34}	c_{23}	c_{12}		
	c_{79}	c_{68}	c_{57}	c_{46}	c_{35}	c_{24}				
	c_{69}	c_{58}	c_{47}							
N								0	0	1

Algorithm 1 Process of VarMSB

Input: Semiprime N and binary variables of p and q
Output: Determined bit and remaining variables of p and q

1: Stage 1: Determine bits of the larger factor q
2: **for** $i = n \to 1$ **do**
3: If the smallest q (with $q_i = 0$) is $< \sqrt{N}$, set $q_i \leftarrow 1$; otherwise stop.
4: **end for**
5: Stage 2: Determine bits of the smaller factor p
6: **for** $i = n \to 1$ **do**
7: If the largest p (with $p_i = 1$) exceeds $\sqrt{N}$, set $p_i \leftarrow 0$; otherwise stop.
8: **end for**
9: **return** Determined bit values of p and q and remaining variables

C. Analytical Variable Reduction for MSB (VarMSB)

The bit pattern in the most significant bits (MSB) defines the numerical range. For instance, any 3-bit number starting with 1 is at least 4, while those starting with 0 are at most 3. Based on this property, we propose Analytical Variable Reduction for MSB (VarMSB), which fixes certain bits by comparing the factors with $\sqrt{N}$. Without loss of generality, let $p \leq q$. The inequality $1 \leq p \leq \sqrt{N} \leq q$ enables analytical determination of several MSB bits in p and q, as outlined in Algorithm 1.

For the larger factor q, we compare the value obtained by assuming all bits of q are 1 except the i-th bit, set to 0, with $\sqrt{N}$. This represents the smallest possible value of q when $q_i = 0$. If this value is still smaller than $\sqrt{N}$, q_i must be set to 1 to satisfy $\sqrt{N} \leq q$; otherwise, q_i remains undetermined. Once $q_i = 1$ is fixed, the same reasoning is applied to q_{i-1}.

The smaller factor p is analyzed similarly. We compare the value obtained by assuming only the most significant and least significant bits of p are 1, with the i-th bit also set to 1, against $\sqrt{N}$. This represents the largest possible value of p when $p_i =$

TABLE IV
MULTIPLICATION TABLE FOR N=4,891

Column	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
p							1	p_5	p_4	p_3	p_2	p_1	1
q							1	q_5	q_4	q_3	q_2	q_1	1
$p \times q$							1	p_5	p_4	p_3	p_2	p_1	1
						q_1	p_5q_1	p_4q_1	p_3q_1	p_2q_1	p_1q_1	q_1	
					q_2	p_5q_2	p_4q_2	p_3q_2	p_2q_2	p_1q_2	q_2		
				q_3	p_5q_3	p_4q_3	p_3q_3	p_2q_3	p_1q_3	q_3			
			q_4	p_5q_4	p_4q_4	p_3q_4	p_2q_4	p_1q_4	q_4				
		q_5	p_5q_5	p_4q_5	p_3q_5	p_2q_5	p_1q_5	q_5					
	1	p_5	p_4	p_3	p_2	p_1	1						
Carries	c_5	c_4	c_3					c_2	c_1				
N	1	0	0	1	1	0	0	0	1	1	0	1	1

1. If this value exceeds $\sqrt{N}$, p_i must be 0 to satisfy $p \leq \sqrt{N}$; otherwise, p_i remains undecided. When $p_i = 0$ is determined, the same analysis continues for the next lower bit p_{i-1}.

D. Special Pattern (SpePattern)

The special pattern (SpePattern) method exploits structural features of the input semiprime to directly determine certain factor variables. This method applies when the following conditions are met:

- The bit width of the semiprime is odd.
- The bits of the semiprime from (MSB-1)-th to the (MSB$-m$)-th, where m is the number of consecutive zeros preceding the first 1, are all 0 ($m \geq 1$).

The first condition ensures that the MSB "1" of the semiprime originates solely from the "1" product term in $p \times q$ and is unaffected by any carries. For example, in the multiplication table for factoring the semiprime 4,891 (Table IV), the bit in column 2^{12} is 1, implying that $c_5 = 0$ and no carry propagates from column 2^{11} to 2^{12}.

The second condition implies that if the bits of the semiprime from (MSB$-$ 1) to (MSB$-m$) are all 0, the corresponding product terms and carries must also be 0; otherwise, a carry would occur or the semiprime bits would no longer be 0. In Table IV, columns 2^{11} and 2^{12} meet this criterion, resulting in p_5, q_5, p_4, q_4, c_5, c_4, and c_3 all being 0.

E. Optimal Block Balancing (OBB)

In practical QA implementations, multiple solutions with the same energy are often returned due to equivalent minima in the cost landscape and inherent randomness in the search process, which may cause the annealer to return different but energetically equivalent solutions across runs.

As the number of variables increases, the explored solution space (candidate count) grows exponentially. If we assume that the QUBO variable count is related to the size of the solution space, the ideal configuration is to keep the variable counts of Blocks I and III as balanced as possible. However, under the MulBlocks method, Blocks I and III are initially partitioned based only on their original variable counts, without fine-tuning. After applying the three reduction techniques (VarLSB, VarMSB, and SpePattern), the variable counts of the two blocks often become highly unbalanced, with Block III usually having far fewer variables than Block I. This imbalance causes the annealer to search a much larger solution space for Block I than for Block III, which may prevent it from returning the correct solution within the same annealing time.

Algorithm 2 Optimal Block Balancing

Input: Initial widths (w_1, w_3) for Blocks I and III, maximum allowable shift w_1, multiplication table of the semiprime.
Output: Optimized widths (w_1^*, w_3^*)

```
Initialize D_min ← ∞, x* ← 0
for x = 1 to w_1 do
    Adjust widths: w_1' ← w_1 − x, w_3' ← w_3 + x
    Recalculate term counts T_1(w_1') and T_3(w_3') by counting non-zero terms in the multiplication table
    Compute imbalance D ← |T_1(w_1') − T_3(w_3')|
    if D < D_min then
        Update D_min ← D, x* ← x
    end if
end for
Set optimized widths: w_1* ← w_1 − x*, w_3* ← w_3 + x*
return (w_1*, w_3*)
```

To address this, we propose the Optimal Block Balancing (OBB) algorithm (Algorithm 2) to determine the optimal value x that minimizes the imbalance in terms between Blocks I and III. Here, x represents the number of columns shifted from Block I to Block III, ensuring that all variables remain covered. In QUBO formulations, each product term in the multiplication table may become a new binary variable (auxiliary variable y) after squaring and reducing higher-order terms to quadratic form. Therefore, balancing the number of terms can be roughly regarded as equivalent to balancing the number of variables.

The algorithm evaluates the number of terms in both blocks for each possible width adjustment x by recalculating the counts of non-zero terms in the multiplication table. Specifically, for a given pair of widths (w_1, w_3), it computes the absolute difference $D = |T_1(w_1) - T_3(w_3)|$, where T_1 and T_3 denote the term counts in Blocks I and III, respectively. The algorithm then iteratively reduces the width of Block I by x and increases the width of Block III by the same amount, selecting the x that minimizes D.

Finally, the block widths are updated to

$$w_1^* = w_1 - x^*, \qquad w_3^* = w_3 + x^*,$$

where x^* is the optimal adjustment value. This procedure effectively balances the variable distribution, preventing Block I or Block III from dominating the problem complexity.

The algorithm evaluates each possible shift x by estimating term counts based on structural reduction rules. For a block of width w, the initial term count is: $T(w) = (w+1)! - 1$.

The VarLSB method analyzes the relationships among LSB variables, allowing certain q_i to be expressed in terms of p_i. In the best case, $p_1 = q_1$, $p_2 = q_2$, and $p_3 = q_3$ hold. The number of reducible terms per column depends on its weight: up to 3 terms are removed when the weight exceeds 2^2, 2

when equal to 2^2, and 1 when equal to 2^1. This maximum reduction is modeled as $R_{max}(w_1)$, defined as $R_{\max}(w_1) = \min(3, \lfloor \log_2(\text{column weight}) \rfloor)$. Accordingly, the total number of terms in Block I after applying VarLSB becomes $T(w_1) = (w+1)! - 1 - \sum_{k=1}^{w_1} R(w_1)$.

For MSB, VarMSB analytically reduces the number of terms according to the number of known bits. Each known bit, either $p_i = 0$ or $q_j = 1$, guarantees a reduction of at least one term of each column, while the simultaneous knowledge of both $p_i = 0$ and $q_j = 1$ leads to an even greater reduction. This minimum guaranteed reduction of each column can be expressed as $L_{\min}(w_3) \geq n$, where n denotes the number of known bits in Block III. The equality holds when the known bits do not appear in any product terms; otherwise, additional reductions occur due to the elimination of those products. Thus, the resulting number of terms in Block III after VarMSB is $T(w_3) = (w+1)! - 1 - \sum_{k=w_3}^{MSB} L(w_3)$.

Moreover, the semiprime contains m consecutive zeros adjacent to the MSB, the SpePattern method analytically reduces the number of terms in Block III. The remaining terms are given by $T_{SpePattern}(w_3, m) = (w_3 - m) \times 2 + \delta$, where δ represents additional terms arising from product terms of unknown variables. Thus, the resulting number of terms in Block III after SP is $T(w_3) = T_{SpePattern}(w_3, m) - \sum_{k=w_3}^{MSB} L(w_3)$.

Finally, based on these analytical results, the OBB algorithm dynamically determines the optimal shift x^*, reducing the width of Block I while increasing that of Block III, thereby minimizing the variable imbalance between the two blocks.

V. Experimental Result

A. Evaluation Model and Experimental Environment

To evaluate the effectiveness of the five proposed methods, we conducted experiments with the following models:

- MulBlocks: The first method proposed partitions the multiplication table into Blocks I and III, omitting the middle Block II, to evaluate the baseline performance.
- MulVar: Builds upon MulBlocks by applying the two variable reduction techniques, VarLSB and VarMSB, to evaluate the effect of variable reduction.
- MulVarSpeObb: Extends MulVar by incorporating the SpePattern and OBB algorithms, which dynamically adjust the widths of Blocks I and III to balance the number of variables between them as evenly as possible.

All three models were implemented in Python 3.8.10 on a computer equipped with an Intel Xeon W-2245 3.90 GHz CPU with 512 GB RAM to construct the QUBOs using the PyQUBO 1.4.0 package [24]. The QUBO models were then solved using on the Amplify AE, GPU-based annealing machine [21]. The obtained solutions were returned to the computer above for verification.

B. Comparison With State-of-the-Art Methods

The performance of MulBlocks, MulVar, and MulVarSpeObb was compared against the state-of-the-art (SOTA) methods of Jiang et al. [17] and Peng et al. [18], where the largest semiprime factored on D-Wave machine was 20 bits. To ensure fair comparison and eliminate machine dependence, these methods were re-implemented on Amplify AE. Semiprime datasets up to 20 bits were taken from SOTAs, while larger instances were randomly generated.

Table V details the semiprime bit width (#Bits), instances (semiprime and factors), evaluated models (two SOTAs and three proposed models), division positions (DivPos), variable counts (#Var), and success rates (SR). For SOTA methods, all block cut points are recorded (e.g., [1,3,5,8]), as they do not omit the middle block. In proposed models, the middle block is omitted; e.g., [1,2) indicates Block I includes column 1, while (5,8] means Block III covers columns 6–8. SOTA variable counts refer to a single model, whereas proposed models list counts separately for Blocks I and III (#VarI and #VarIII). Annealing time was fixed at 10 s per block for proposed models and 20 s for SOTAs [17], [18], providing equvalent total time since MulBlocks performs two anneals. The success rate denotes the probability of correctly obtaining prime factors, recognizing that some trials may fail. Each experiment was repeated 100 times.

When converting the objective function into a QUBO, higher-order terms must be reduced to quadratic forms. As noted in Section II-C, we apply penalty terms and auxiliary variables. Two penalty schemes were used: PA-1 and PA-2. In PA-1, $P = 2$ is used when reducing a cubic term, and $P^2 = 4$ for a quadratic term. The same rules apply to higher-order terms. The final penalty value is obtained by adding the penalty values of all higher-order terms. The penalty value for PA-2 is twice as that of PA-1. The reported success rate corresponds to the best result over 100 trials for each instance.

On Amplify AE, replicated SOTA methods factored semiprimes up to 26 bits (versus 20 bits on D-Wave), though smaller cases occasionally failed. This improvement reflects Amplify AE's fully connected architecture, unlike the D-Wave 2000's 6-node connectivity. As bit width grows, variable counts rise sharply. MulBlocks, MulVar, and MulVarSpeObb significantly reduce variables compared to SOTAs, with greater gains for larger instances. Without middle-block omission, higher-order terms proliferate, introducing many auxiliary variables. For the 19-bit semiprime 376,289, SOTA methods require about 90 variables, compared with 19 for MulBlocks, 12 for MulVar, and 11 for MulVarSpeObb.

Across the proposed methods (VarLSB, VarMSB, and SpePattern), clear improvements are observed. For the 19-bit semiprime, the last three bits $(001)_2$ allow VarLSB to directly deduce relationships among p_1, p_2, p_3 and q_1, q_2, q_3, reducing variables from weights 2^1 to 2^5 by (1,2,3,3,3), totaling 12 in Block I. In VarMSB, knowing two variable values (p_{10}, p_{11}) in advance transforms product terms into single variables, reducing seven in Block III. With SpePattern, three variables (p_{10}, p_{11}, q_{11}) become known; after rebalancing via OBB, the variables in both blocks are further reduced. Overall, while SOTA success rates drop sharply beyond 20 bits, the proposed three models maintain 100%.

TABLE V
COMPARISON OF MULBLOCKS, MULVAR, MULVARSPEOBB AND SOTAS

#Bits	Instance	Model	DivPos	#Var (#VarI, #VarIII)	SR (%)
8	143 =11×13	Jiang	[1,3,5,8]	12	100
		Peng		5	100
		MulBlocks	[1,2) (5,8]	(3,4)	100
		MulVar		(3,4)	100
		MulVarSpeObb	[1,2) (5,8]	(3,4)	100
9	323 =17×19	Jiang	[1,4,6,9]	20	100
		Peng		13	100
		MulBlocks	[1,3) (7,9]	(7,4)	100
		MulVar		(5,4)	100
		MulVarSpeObb	[1,2) (6,9]	(3,4)	100
11	1591 =37×43	Jiang	[1,4,7,11]	29	100
		Peng		22	100
		MulBlocks	[1,3) (8,11]	(7,8)	100
		MulVar		(6,7)	100
		MulVarSpeObb	[1, 3) (8,11]	(6,7)	100
16	59989 =239×251	Jiang	[1,4,7,10,13,16]	59	100
		Peng		52	100
		MulBlocks	[1,4) (11,16]	(11,13)	100
		MulVar		(7,9)	100
		MulVarSpeObb	[1, 4) (11, 16]	(7,9)	100
19	376289 =571×659	Jiang	[1,5,8,11, 14,17,19]	95	75
		Peng		90	94
		MulBlocks	[1,5) (14,19]	(19,19)	100
		MulVar		(12,12)	100
		MulVarSpeObb	[1,4) (13,19]	(7,11)	100
20	1005973 =997×1009	Jiang	[1,4,7,10, 13,16,20]	96	97
		Peng(2019)		89	100
		MulBlocks	[1,5) (14,20]	(19,19)	100
		MulVar		(11,11)	100
		MulVarSpeObb	[1,5) (14,20]	(11,11)	100
24	10934491 =3163×3457	Jiang	[1,4,7,10,13, 16,19,22,24]	142	2
		Peng		135	0
		MulBlocks	[1,6) (17,24]	(24,24)	100
		MulVar		(20,20)	100
		MulVarSpeObb	[1,6) (17,24]	(20, 20)	100
25	21364867 =4111×5197	Jiang	[1,4,7,10,13, 16,19,22,25]	167	0
		Peng		162	2
		MulBlocks	[1,4,7) (19,25]	(35,24)	100
		MulVar		(23,14)	100
		MulVarSpeObb	[1,6) (18,25]	(15,16)	100
26	51880877 =6911×7507	Jiang	[1,4,7,10,13, 16,19,22,26]	167	1
		Peng		159	4
		MulBlocks	[1,7) (19,25]	(35,24)	100
		MulVar		(23,17)	100
		MulVarSpeObb	[1,7) (18,25]	(23,17)	100

TABLE VI
COMPARISON BETWEEN THREE PROPOSED MODELS MULBLOCKS, MULVAR, AND MULVARSPEOBB

#Bits	Instance	DivPos	Model	#VarI	#VarIII	#CanI	#CanIII	VerTime(s)	SR (%)
29	299427217 = 16553 ×18089	[1, 4 ,8) (22, 29]	MulBlocks	47	36	128	36	0.005	100
			MulVar	41	17	128	26	0.005	100
		[1, 5) (19, 29]	MulVarSpeObb	20	18	16	436	0.009	100
34	13098603389 =112559 ×116371	[1, 5, 9) (25, 34]	MulBlocks	62	52	256	570	0.015	100
			MulVar	54	42	256	569	0.014	100
		[1, 4, 8) (24, 28, 34]	MulVarSpeObb	37	54	288	1142	0.026	100
40	723320469463 =889727 ×812969	[1, 5, 10) (29, 33, 40]	MulBlocks	77	80	512	3413	0.224	100
			MulVar	74	68	512	2343	0.111	100
		[1, 5, 10) (29, 33, 40]	MulVarSpeObb	76	69	1152	2343	0.213	100
42	4216034734687 =2043091 ×2063557	[1, 6, 11) (31, 35, 42]	MulBlocks	92	80	970	356	0.053	97
			MulVar	92	35	1024	260	0.052	100
		[1, 5, 9) (29, 33, 42]	MulVarSpeObb	56	61	576	1043	0.084	100
44	17586909926101 =4193393 ×4193957	[1, 6, 11) (32, 36, 44]	MulBlocks	92	94	963	13	0.036	99
			MulVar	84	18	1024	4	0.028	100
		[1, 4, 7) (28, 33, 44]	MulVarSpeObb	24	41	96	126	0.026	100
46	66463993662463 =8178997 ×8126179	[1, 6, 12) (34, 38, 46]	MulBlocks	115	94	802	903	0.127	65
			MulVar	114	46	1925	518	0.271	100
		[1, 5, 10) (32, 36, 46]	MulVarSpeObb	76	74	1152	2081	0.201	100
48	278377626385153 =16620301 ×16749253	[1, 6, 12) (35, 39, 48]	MulBlocks	115	110	823	371	0.067	60
			MulVar	115	37	1827	262	0.104	87
		[1, 5, 10) (33, 37, 48]	MulVarSpeObb	75	65	512	1027	0.056	100
51	1495000453129013 =33554467 ×44554439	[1, 7, 13) (38, 42, 52]	MulBlocks	133	131	678	9114	1.802	19
			MulVar	136	95	2489	9260	2.588	36
		[1, 6, 12) (37, 41, 51]	MulVarSpeObb	104	113	2478	8449	0.990	71
54	17998604003308351 =134156371 ×134161381	[1, 7, 14) (40, 44, 54]	MulBlocks	156	131	370	65	0.288	0
			MulVar	168	23	1389	33	0.517	15
		[1, 5, 9) (35, 40, 54]	MulVarSpeObb	56	73	576	2058	0.142	100
57	72057876968901713 =268435459 ×268436507	[1, 8, 15) (43, 47, 57]	MulBlocks	179	151	104	7	0.009	0
			MulVar	198	21	814	1	0.009	2
		[1, 5) (33, 57]	MulVarSpeObb	20	13	16	65	0.002	100

C. Comparison Between Proposed Models

The three proposed models (MulBlocks, MulVar, and MulVarSpeObb) were further applied to semiprimes larger than 26 bits. Table VI summarizes the bit width (#Bits), instance count (Instance), division positions (DivPos), variable counts per block (#VarI, #VarIII), candidate counts per block (#CanI, #CanIII), verification time (VerTime), and success rate (SR) over 100 trials. The candidate count denotes the number of solutions satisfying each block's constraints; since each block uses partial multiplication information, multiple valid solutions may exist. Piece weights (σ_1 and σ_2 in Section IV-A) were set to 1, with a 10 s annealing time per run.

In MulBlocks, variable counts for Blocks I and III are similar, while MulVar may show imbalance depending on the effectiveness of VarLSB or VarMSB. VarLSB can deduce q_3, q_2, q_1, whereas VarMSB can directly fix variables to 0 or 1, eliminating more terms. For example, in the 34-bit semiprime 13,098,603,389, VarLSB in MulVar reduces 8 Block I variables compared to MulBlocks. In contrust, for the 57-bit 72,057,876,968,901,713, its performance degrades due to redundant auxiliary variables generated by PyQUBO during asymmetric reduction. For example, given $p_1p_2q_3q_4$, auxiliary variables p_1q_3 and p_2q_4 already exist, so no additional one is needed. However, PyQUBO may still generate an unnecessary auxiliary variable, such as $p_2q_3q_4$. VarMSB's efficiency depends heavily on the bit patterns—more deducible bits yield smaller QUBOs. For the 48-bit semiprime 278,377,626,385,153, it cuts variable counts by 66% versus MulBlocks. MulVarSpeObb balances variables between Blocks I and III, simplifying both and improves the success rates.

PA-1 and PA-2 were both tested, selecting the configuration yielding the largest number candidate count for a given semiprime. The relationship between success rate and candidate count indicates that QA effectively narrows the original search space, making SR proportion the explored region. For the 29-, 34-, and 40-bit semiprimes, both MulBlocks and MulVar achieved 100% SR. In Block I, candidate counts are identical because MulVar's reduction of q_1 and q_2 preserves equivalent constraints to MulBlocks. In contrast, for Block II, VarMSB directly fixes variable values, shrinking the search space.

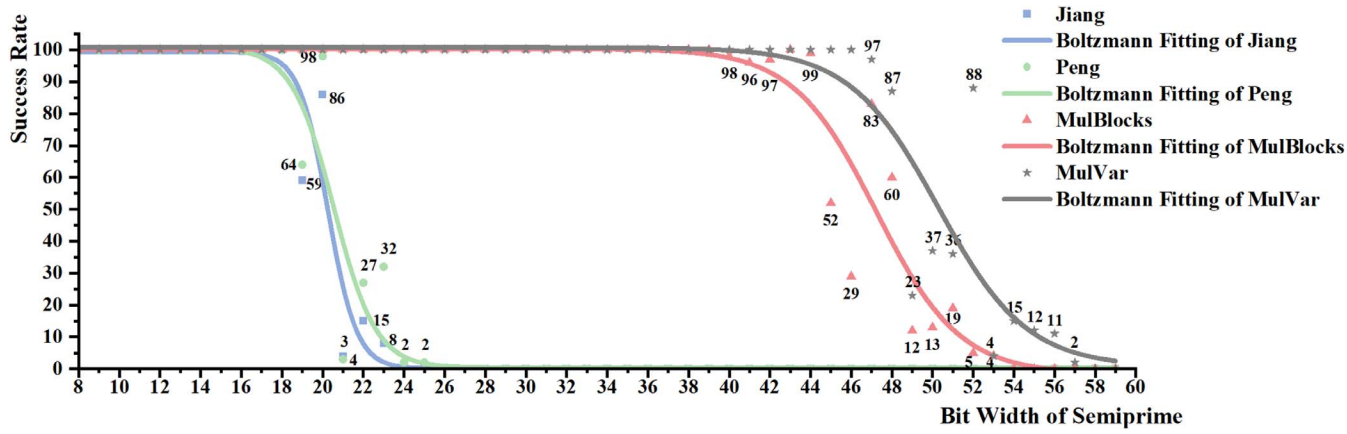

Fig. 1. Success rate of MulBlocks, MulVar, and SOTA methods.

Generally, more variables produce larger candidate counts, though cases with counts, e.g., such as the 40-bit instance, show higher counts in Block III due to a larger maximum Hamiltonian energy range.

From 42 bits onward, the success rate of MulBlocks begins to decline. Even at the candidate generation stage in Block I, the solution space (963) no longer covers the full space (1024), reflecting the increased number of variables and combinations. As problem size grows, selecting an appropriate penalty values becomes more challenging: a higher variable count not only increases the Hamiltonian's maximum energy but also reduces the annealer's ability to distinguish valid from invalid candidates in Blocks I and III. In particular, an improperly tuned penalty may suppress useful structure in these blocks, preventing the annealer from reaching desirable solutions—even when they lie within feasible energy bounds. Effective discrimination thus demands finer penalty tuning. This explains why, for larger instances (e.g., 54 and 57 bits), the accessible search space can shrink dramatically—testing only a few penalty values is insufficient to find a range that enhances annealing precision. Nevertheless, favorable cases do arise; for instance, in the 54-bit case, a suitable penalty value happened to be found, enabling broader candidate exploration. A discussion of penalty value effects is provided in Section V-E.

The total runtime of QA-based factorization consists of two components: the fixed annealing time and the verification time. The latter scales approximately with the number of candidates, since verification simply multiplies factor pairs. More specifically, runtime complexity follows $O(n_{\#\text{CandidateI}} \times n_{\#\text{CandidateIII}})$. However, models with higher success rates often terminate verification earlier once valid factors are found, making overall runtime primarily governed by the number of candidates rather than by the annealing duration.

Fig. 1 shows the success rate as a function of semiprime bit width. Boltzmann fitting curves are shown for SOTA methods, MulBlocks and MulVar; MulVarSpeObb is excluded since its trend is determined more on semiprime structure than on bit width. In terms of success rate, MulBlocks can stably factors numbers up to 40-bit numbers, while MulVar extend this range to 47 bits.

TABLE VII
EXTREME RESULT OF MULVARSPEOBB

#Bit	DivPos	#KnowVar	#VarI	#VarIII	#CanI	#CanIII	SR (%)
61	[1, 2) (33, 61]	$\{p_i\}_{i=4}^{29}$, $\{q_i\}_{i=4}^{29}$	3	2	5	5	100
101	[1, 4) (54, 101]	$\{p_i\}_{i=8}^{49}$, $\{q_i\}_{i=8}^{49}$	12	9	8	19	100
389	[1, 4) (198, 389]	$\{p_i\}_{i=7}^{193}$, $\{q_i\}_{i=7}^{193}$	12	7	8	11	100
415	[1, 6) (213, 415]	$\{p_i\}_{i=17}^{206}$, $\{q_i\}_{i=17}^{206}$	29	23	32	2558	100
655	[1, 6) (333, 655]	$\{p_i\}_{i=17}^{326}$, $\{q_i\}_{i=17}^{326}$	29	32	32	2563	100
1185	[1, 6) (598, 1185]	$\{p_i\}_{i=18}^{591}$, $\{q_i\}_{i=18}^{591}$	29	25	32	5693	100
2049(1)	[1, 5) (1029, 2049]	$\{p_i\}_{i=12}^{1023}$, $\{q_i\}_{i=12}^{1023}$	20	15	16	173	100
2049(2)	[1, 8) (1032, 2049]	$\{p_i\}_{i=23}^{1023}$, $\{q_i\}_{i=23}^{1023}$	53	31	128	22356	100
2049(3)	[1, 8) (1032, 2049]	$\{p_i\}_{i=27}^{1023}$, $\{q_i\}_{i=27}^{1023}$	53	39	128	128	0

D. Extreme Performance of MulVarSpeObb

The performance of MulVarSpeObb is summarized in Table VII, showing semiprime bit width (#Bits), division position (DivPos), known variables from the MulVarSpeObb analysis (#KnowVar), variable counts per block (#VarI and #VarIII), candidate counts (#CanI and #CanIII), and success rates (SR) over 100 trials, with 10 s per anneal. All piece coefficients are set to 1, and PA-1 (Sections II-C and V-B), is used as the penalty algorithm for higher-order term reduction.

Although 2048-bit RSA moduli are widely regarded as current standard, the benchmark set includes smaller semiprimes and instances around this size. As the current implementation of SpePattern supports only odd bit widths, 2049-bit semiprimes are used instead. The benchmark set includes the 61-, 101-, 389-, 415-, 655-, and 1185-bit cases, as well as three distinct 2049-bit semiprimes; detailed semiprime data appear in the final section.

From Table VII, MulVarSpeObb successfully factors semiprimes with odd bit widths up to 2049 bits and, in principle, beyond. It substantially reduced variable counts in both Blocks I and III. In Block III, quadratic terms and multiplication-table variables are eliminated without introducing auxiliary variables, and this reduction persists even as Block III widens under OBB adjustment. Block I similarly benefits from narrower width and the removal of redundant quadratic terms.

For the 2049-bit instance, the method tolerate the highest "1" (excluding the MSB) up to position 2^{1046} from the LSB. If

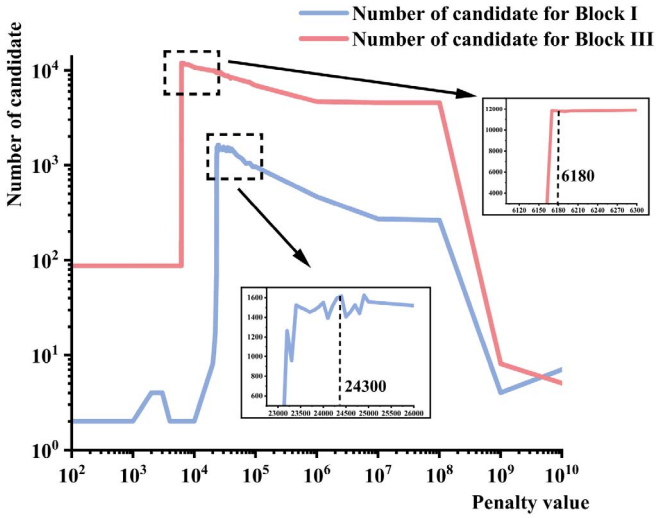

Fig. 2. Number of candidates with different penalty value.

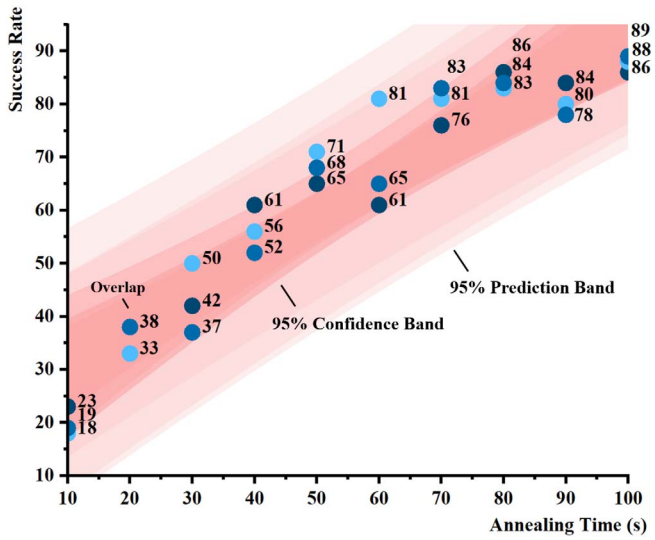

Fig. 3. Success rate variation with different annealing times. (Each dot shows the successes over 100-runs.)

the next-highest "1" appears beyond this position, the Block III Hamiltonian becomes excessively large, making penalty tuning ineffective at separating local from global minima, causing factorization to fail, as seen in the "2049-bit(3)" case.

Although MulVarSpeObb is highly specialized, its underlying idea is extensible. Adjusting multiplication-table width to control the maximum Hamiltonian energy could facilitate the selection of suitable penalty values while retaining critical structural information. Likewise, exploiting consecutive zero patterns, not only from the MSB but elsewhere, could yield additional known variables, potentially benefiting the factorization of even-length semiprimes. From an engineering perspective, QUBO is well suited to such structure-aware strategies, leveraging regularities in the multiplication table to reduce the effective search space and improve computational efficiency.

E. Aberration Study

This section evaluates the performance of the MulBlocks model under various conditions, including different penalty values, annealing times, numbers of pieces, and piece coefficients, to examine their impact on success rate. Both the number of pieces and their coefficients alter the maximum Hamiltonian energy and thus the choice of penalty value. These two factors are therefore interdependent. For each configuration, PA-1 is applied for given semiprime to ensure a consistent comparison.

1) Penalty Value: To evaluate the impact of penalty magnitude on the number of valid candidates, experiments were conducted on a 51-bit semiprime 1,495,000,453,129,013. Specifically, the relationship between penalty value and candidate count found in Blocks I and III was analyzed by varying the penalty logarithmically from 10^0 to 10^{10}. The penalty yielding the largest number of candidates was identified as optimal. Fig. 2 shows that suitable penalties substantially increase candidate counts, thereby improving success probability. The annealing time was fixed at 10 s per block.

As shown in Fig. 2, achieving the maximum number of candidates requires the penalty value to lie within a specific range. If the penalty is even slightly below this optimal window, the number of candidates drops drastically. For instance, the optimal range for Block I is approximately [23,400, 37,000], while for Block III it is around [6,170, 10,000]. Notably, when the penalty for Block III is set to 6,160, only 87 candidates are found, whereas increasing it slightly to 6,170 yields 11,851 candidates. This sharp contrast highlights the importance of careful penalty tuning to maximize candidate generation and, consequently, the overall success rate.

However, the precise relationship between the optimal penalty value and properties of the QUBO remains unclear. Nevertheless, by setting the penalty values to 23,400 for Block I and 6,180 for Block III, the success rate improves dramatically from 19% (as reported in Table VI) to 81%. This demonstrates that carefully selecting penalty values plays a crucial role in maximizing the success rate, thereby greatly enhancing the ability to factor semiprimes.

2) Annealing Time: With well-chosen penalties, increasing annealing time enables broader exploration, raising the likelihood of correct factorization. As an example, for a 51-bit semiprime, 1,495,000,453,129,013, success rates were measured for three times for each annealing duration (10–100 s, in 10 s increments) using PA-1.

Fig. 3 presents the success rate as a function of annealing time, including 95% prediction and confidence bands. Fig. 4 illustrates the corresponding trend in the candidate counts.

Both success rate and candidate count increase with annealing time and eventually saturate, indicating more thorough exploration of the solution space. The increased number of candidates enhances the probability of correct factorization. Therefore, even when penalties are suboptimal, extending annealing time can enhance the annealer's ability to discriminate valid from invalid solutions.

3) Different Numbers of Pieces: To minimize the generation of higher-order terms in the objective function, Blocks I and III are subdivided into pieces. Table VIII lists the partitioning

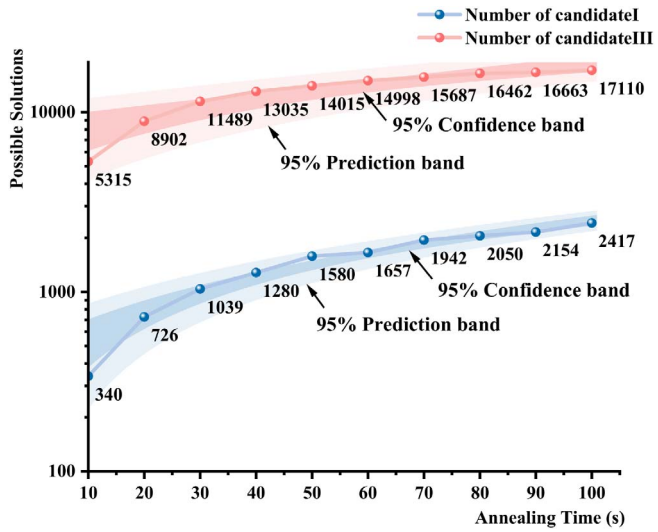

Fig. 4. Number of candidates obtained in Blocks I and III.

TABLE VIII
PERFORMANCE COMPARISON OF TWO AND THREE PIECES

Instance	#Bit	#Piece	DivPos	#VarI	#VarIII	SR(%)
66463993662463= 8178997×8126179	46	2	[1, 6, 12) (34, 38, 46]	115	94	65
		3	[1, 4, 8, 12) (34, 38, 42, 46]	108	96	29
87146229836093 = 8388617×10388629	47	2	[1, 6, 12) (35, 39, 47]	115	110	60
		3	[1, 4, 8, 12) 35, 39, 43, 47]	108	111	49
1495000453129013 =33554467×44554439	51	2	[1, 7, 13) (38, 42, 51]	133	131	19
		3	[1, 5, 9, 13) (38, 42, 46, 51]	126	127	4
6516867607983367 =67108879×97108873	53	2	[1, 7, 14) (40, 44, 53]	156	131	2
		3	[1, 6, 10, 14) (40, 44, 48, 53]	143	127	1
172562341101157297 =268435459×268436507	58	2	[1, 8, 15) (43, 47, 58]	179	151	0
		3	[1, 7, 11, 15) (43, 47, 51, 58]	162	145	0

of five large semiprimes. Blocks I and III are divided into two or three pieces simultaneously. Constraints are evaluated on each piece, and a QUBO is constructed by adding the QUBOs of all pieces. Table VIII presents the data, including the bit width of the semiprime (#Bits), the number of pieces in both Blocks I and III (#Piece), the position of the division into blocks and subdivision into pieces (DivPos), the maximum number of variables (#Variables), the number of variables for each block (Blocks I and III), and the success rate (SR) in 100 trials for 10s annealing time for each separate annealing. The coefficients of pieces are all set to 1.

The success rate decreased for all semiprimes when divided into three pieces. This is explained by the omission of the product relationships between some variables, which resulted in the loss of information. To illustrate this, we consider the energy of two adjacent columns, denoted as x and y, ordered from MSB to LSB. In the case of two pieces, the constraint is given by $(2x + y)^2$. In the case of three pieces, this becomes $x^2 + y^2$, losing information xy. Consequently, the number of possible solutions that are not correct increases rapidly. The number of variables may remain the same or even increase for three pieces owing to changes in the generation of auxiliary variables, making the problem difficult to solve.

4) Coefficients for the Pieces: Adjusting the coefficients when adding different pieces changes the Hamiltonian, thereby changing the probability of finding the correct solution, and the success rate. Therefore, different coefficients are explored to identify suitable values. In an example of a 40-bit semiprime 723,320,469,463, the coefficients of Block III are initially set to 1, serving as a baseline. The objective function of Block I is $f_1 = \sigma_1 \times f_{11} + \sigma_2 \times f_{12}$. The annealing time is 1 s for each separate annealing. The maximum energy of the objective function, as indicated by [25], is determined by setting all positive terms to 0 and negative terms to 1. It can be calculated that the energies of f_{11} and f_{12} are 15129 and 260100, respectively.

The variations of σ_1/σ_2 and σ_2/σ_1 are tested from 1 to 100,000. We found that the success rate when σ_1 is larger is generally slightly higher than when σ_2 is larger. However, it fluctuates in the range of approximately 40% to 60%, indicating that, within a certain range, assigning coefficients to functions does not significantly affect the results. Assigning higher coefficients to functions with lower energy levels increases the success rate to certain extent but not significantly. When σ_1/σ_2 is greater than approximately 77,842 or σ_2/σ_1 is greater than approximately 3,149, the success rate instantly drops to zero. This is because assigning excessively large coefficients to one function causes the exploration space to ignore the other function.

F. Discussion

In the experiments conducted in this study, several challenges were encountered when solving large semiprimes using QA. First, parameter tuning—particularly the choice of penalty values—plays a critical role in determining the success probability. Adjusting either the penalty magnitude or the ratio and number of subproblems ("pieces") effectively modifies the QUBO formulation, thereby changing the distinguishability between the global and local minima. Unfortunately, there is currently no established framework for systematically determining appropriate parameter values; they must be empirically optimized for each problem instance. This lack of standardization remains a major obstacle for QA-based methodology.

Second, longer annealing times generally increase the success rate by enabling broader exploration of the energy landscape. However, if the penalty parameters are poorly tuned, the system may still converge to suboptimal configurations, regardless of annealing duration, since the underlying QUBO does not correctly represent the problem constraints.

Overall, this study adopts a pragmatic, hybrid approach: QA is used to generate a biased set of high-quality candidate solutions, which are then refined using post-processing filters to

identify the exact factors. Although these experiments remain constrained by current hardware scale, they provide valuable insights into how QA can be further scaled and tuned—through improved QUBO design, weight calibration, and parameter optimization—to address larger instances in the future.

VI. Conclusion

This study proposed five methods for factoring large semiprimes using the QUBO model: MulBlocks, VarLSB, VarMSB, SpePattern, and OBB. MulBlocks divides the original problem into subproblems through multiple annealing rounds. VarLSB and VarMSB reduce the number of variables through preprocessing, while SpePattern identifies fixed variables based on specific bit patterns in odd-width semiprimes. OBB balances the variable distribution across blocks. The MulBlocks model applies only MulBlocks, MulVar combines the first three, and MulVarSpeObb integrates all five techniques. The novelty lies in effectively reducing problem complexity to overcome current hardware limitations.

Experimental results using Amplify AE demonstrate that proposed MulBlocks and MulVar frameworks substantially outperform state-of-the-art methods, both in terms of variable reduction and factorable bit width. Notably, MulBlocks achieves a 100% success rate up to 40-bit semiprimes, and MulVar extends this to 47 bits. Under certain structural conditions, MulVarSpeObb successfully factors semiprimes up to 2049 bits, showing remarkable scalability.

More importantly, this work highlights the practical use of quantum annealing as an engineering tool for solving structured combinatorial problems like integer factorization. Through penalty tuning, sub-problem decomposition, and structure-based preprocessing, we make full use of the underlying multiplication table structure to guide QUBO modeling. Extensive analysis of the relationship between penalty value, search space, and annealing configuration supports the robustness and generality of our methods.

Finally, although experiments are conducted on Amplify AE, the techniques are platform-agnostic, applicable across a wide range of quantum and quantum-inspired annealers, demonstrating the engineering potential of QA beyond academic benchmarks.

Large Semiprime Instances

61-bit: 1152921515344265237=1073741827×1073741831

81-bit: 1208925819660808663073173=1099511627791×1099511627803

101-bit: 1267650600228402790082356974917=1125899906842723×1125899906842679

389-bit: 6304320991423116673964646416022978208812758283274471466895328847204369403111576266057543426506131359000245248217597 61=251084069415467230553431576928306656644094217778561380516111×251084069415467230553431576928306656644094217778561380516511

415-bit: 4230758200257591033292257971409734654901789970971399803423464642936159443950010128086809689683425482534758701005446785748090 9=20568806966515075526937114781966881312284198320419748291860134 3

655-bit: 7475102707912204646221695558779357306705065586276040590260949021326172433954697030051287550646764929453786135682173621498582497952348659586449527763962581118683522541817232192503746288861278056363=2734063405978764905465627783897026706691461788616515545532213258012441248999219904029391471279396811×2734063405978764905465627783897026706691461788616515545532213258012441248999219904029391471279076 23

1185-bit: 2627331460019798373118191176765844037411487540504255664326970949768179426764182376006696248967902553296870562596429685636685932508974710239409120160328478598733444743599852153687094925433885802804649497283614495317678671703623614972071142396037932551044940384539613237071892315104998937337858995357229063361292110767525724264647830541435971186792349984970 53=1620904519094137874418909321754359824614236809469701914060803644410411254458167244687385565994962419643827299457539370774373105888832724729643310482075767065258274141953714673308 9=2056880696651507552693711478196688131228419832041974829186341633×1007741827×1007741831

2049-bit(1): 3231700607131100730071487668866995196044410266971548403213034542752465513886789089319720141152291346368871796092189801949411955915049092109508815238644828312063087736730099609175019775038965210679605763838406756827679221864261975616183809433847617047058164585203630504288757589154106580860755239912393038601735856135772260659088148536801978607997063143182601350365110933004228071266911470789482209341934639093297745664402195106522295400483385749469121645562397401635543996364009763121407432276116671668237515852693259685298927780418069998032005781690930295773760958215306755393992828505505028431979789805542131975661 1=1797693134862315907729305190789024733617976978942306572734300811577326758055009631327084773224075360211201138798713933576587897688144166224928474306394741243777678934248654852763022196012460941194530829520850057688381506823424628814739131105408272371633505106845862982399472459384797163048353563296242241391293×1797693134862315907729305190789024733617976978942306572734300811577326758055009631327084773224075360211201138798713933576587897688144166224928474306394741243777678934248654852763022196012460941194530829520850057688381506823424628814739131105408272371633505106845862982399472459384797163048353563296242241378 59

2049-bit(2): 3231700607131100730071487668866995196044410266971548403213034542752465513886789089319720141152291346368871796092189801949411955915049092109508815238644828312063087736730099609175019775038

96521067960576383840675682767922186426197561618380
94338476170470581645852036305042887575891541065808
60755239912393182290952633191013055962751601161575
35125150103506637965255702743413018498752141464132
57834473595761797411538576763333959063903351712722
98027712936167429823169477881669125811687043877669
82515640494345464433490321727932769539260984581375
75386522217117750400775800245374000890997875142780
92107461656274524320009541688 9=17976931348623159
07729305190789024733617976978942306572734300811577
32675805500963132708477322407536021120113879871393
35765878976881441662249284743063947412437776789342
48654852763022196012460941194530829520850057688381
50682342462881473913110540827237163350510684586298
23994724593847971630483535632962422813508 3×179769
313 48623159077293051907890247336179769789423065727
343 00811577326758055009631327084773224075360211201
138 79871393357658789768814416622492847430639474124
377 76789342486548527630221960124609411945308295208
500 57688381506823424628814739131105408272371633505
106 84586298239947245938479716304835356329624228135
083

2049-bit(3): 32317006071311007300714876688669951960
44410266971548403213034542752465513886789089319720
14115229134636887179609218980194941195591504909210
95088152386448283120630877367300996091750197750389
65210679605763838406756827679221864261975616183809
43384761704705816458520363050428875758915410658086
07552399123949919249765129282762318146154311718795
18328513183680593027193794664229483660630158642959
70494401214111908756242867851648204740087964530604
40386786445865091125003510438098978745247680065357
47507407608364075344484077846678536850745009514182
96416719166731354128073729406932207287106818145472
84636257601421928076399999231=1797693134862315907
72930519078902473361797697894230657273430081157732
67580550096313270847732240753602112011387987139335
76587897688144166224928474306394741243777678934248
65485276302219601246094119453082952085005768838150
68234246288147391311054082723716335051068458629823
99472459384797163048353563296242952453 13×179769313
48623159077293051907890247336179769789423065727343
00811577326758055009631327084773224075360211201138
79871393357658789768814416622492847430639474124377
76789342486548527630221960124609411945308295208500
57688381506823424628814739131105408272371635051068
45862982399472459384797163048353563296242616890 87

REFERENCES

[1] S. S. Wagstaff, "The joy of factoring," *Amer. Math. Soc*, vol. 68, 2013.

[2] R. Dridi and H. Alghassi, "Prime factorization using quantum annealing and computational algebraic geometry," *Sci. Rep.*, vol. 7, no. 1, 2017, Art. no. 43048.

[3] A. K. Lenstra, H. W. Lenstra, Jr, M. S. Manasse, and J. M. Pollard, "The number field sieve," in *Proc. 22nd Annu. ACM Symp. Theory Comput.*, 1990, pp. 564–572.

[4] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. New York, NY, USA: Cambridge Univ. Press, 2002.

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, 1999.

[6] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'brien, "Experimental realization of Shor's quantum factoring algorithm using qubit recycling," *Nat. Photon.*, vol. 6, no. 11, pp. 773–776, 2012.

[7] J. A. Smolin, G. Smith, and A. Vargo, "Oversimplifying quantum factoring," *Nature*, vol. 499, no. 7457, pp. 163–165, 2013.

[8] E. Lucero et al., "Computing prime factors with a Josephson phase qubit quantum processor," *Nat. Phys.*, vol. 8, no. 10, pp. 719–723, 2012.

[9] U. Skosana and M. Tame, "Demonstration of Shor's factoring algorithm for n= 21 on IBM quantum processors," *Sci. Rep.*, vol. 11, no. 1, 2021, Art. no. 16599.

[10] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Rev. Mod. Phys.*, vol. 90, no. 1, 2018, Art. no. 015002.

[11] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution," 2000, *arXiv: quant-ph/0001106*.

[12] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, "Adiabatic quantum computation is equivalent to standard quantum computation," *SIAM Rev.*, vol. 50, no. 4, pp. 755–787, 2008.

[13] X. Peng et al., "Quantum adiabatic algorithm for factorization and its experimental implementation," *Phys. Rev. Lett.*, vol. 101, no. 22, 2008, Art. no. 220405.

[14] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, "Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system," *Phys. Rev. Lett.*, vol. 108, no. 13, 2012, Art. no. 130501.

[15] S. Pal, S. Moitra, V. Anjusha, A. Kumar, and T. Mahesh, "Hybrid scheme for factorisation: Factoring 551 using a 3-qubit NMR quantum adiabatic processor," *Pramana*, vol. 92, no. 2, p. 26, 2019.

[16] D-Wave, "Wave systems: The practical quantum computing company," 2023. Accessed: Mar. 5, 2023. [Online]. Available: https://www.dwavesys.com/

[17] S. Jiang, K. A. Britt, A. J. McCaskey, T. S. Humble, and S. Kais, "Quantum annealing for prime factorization," *Sci. Rep.*, vol. 8, no. 1, 2018, Art. no. 17667.

[18] W. Peng et al., "Factoring larger integers with fewer qubits via quantum annealing with optimized parameters," *Science China Phys., Mechan. Astron.*, vol. 62, pp. 1–8, 2019.

[19] B. Wang, X. Yang, and D. Zhang, "Research on quantum annealing integer factorization based on different columns," *Front. Phys.*, vol. 10, p. 518, 2022, Art. no. 914578.

[20] X. Guo, G. Miao, S. Nishizawa, and S. Kimura, "Prime factorization based on multiple quantum annealings on partial constraints with analytical variable reduction" in *Proc. IEEE Int. System-on-Chip Conf.*, 2023, pp. 1–6.

[21] Fixstars Amplify "The quantum computing cloud—Fixstars Amplify," 2024. Accessed: Apr. 19, 2024. [Online]. Available: https://amplify.fixstars.com/

[22] D-Wave Leap "New QPU solver: Advantage performance update," 2023. Accessed: Mar. 5, 2023. [Online]. Available: https://support.dwavesys.com

[23] F. Glover, G. Kochenberger, R. Hennig, and Y. Du, "Quantum bridge analytics I: A tutorial on formulating and using QUBO models," *Ann. Operat. Res.*, vol. 314, no. 1, pp. 141–183, 2022.

[24] "Pyqubo," 2024. Accessed: Apr. 19, 2024. [Online]. Available: https://pypi.org/project/pyqubo/

[25] R. Tanburn, O. Lunt, and N. S. Dattani, "Crushing runtimes in adiabatic quantum computation with energy landscape manipulation (ELM): Application to quantum factoring," 2015, *arXiv:1510.07420*.

Xinyi Guo received the B.E. degree from Sichuan University, in 2022, and the M.E. degree from Waseda University, in 2023. She is currently working toward the Ph.D. degree with Kyoto University. Her research interests include quantum computing, Ising machine, quantum error correction, and combinatorial optimization problems.

Geguang Miao received the B.E. degree from Xidian University, in 2020, and the M.E. degree from Waseda University, in 2021. His research interests include quantum annealing and combinatorial optimization problems.

Shinichi Nishizawa received the B.E. degree from Ritsumeikan University, in 2009, and the M.E. and Ph.D. degrees from Kyoto University, in 2011 and 2015, respectively. From 2022 to 2024, he was an Assistant Professor with Waseda University. He is currently an Associate Professor with the Graduate School of Advanced Science and Engineering, Hiroshima University. His research interests include CAD for low-power and low-voltage digital circuits.

Shinji Kimura (Life Member, IEEE) received the B.E., M.E., and Ph.D.Eng. degrees in information science from Kyoto University, Kyoto, Japan, in 1982, 1984, and 1989, respectively. Since 1985, he has been an Assistant Professor with Kobe University. Since 1993, he has been an Associate Professor with Nara Institute of Science and Technology. Since 2002, he has been a Professor with Waseda University. He has served an Executive Committee Member of ICCAD 2011 and 2012 and a General Chair of ASPDAC 2013. Since 2015, he has been a fellow of IEICE. He is a member of the Information Processing Society of Japan.

Takashi Sato (Senior Member, IEEE) received the B.E. and M.E. degrees from Waseda University, Tokyo, Japan, and the Ph.D. degree from Kyoto University, Kyoto, Japan. From 1991 to 2003, he was with the Hitachi Ltd., Tokyo. From 2003 to 2006, he was with the Renesas Technology Corp., Tokyo. He was with Tokyo Institute of Technology, Yokohama, Japan. In 2009, he joined the Graduate School of Informatics, Kyoto University, where he is currently a Professor. He received the Beatrice Winner Award at ISSCC 2000, and the Best Paper Award at ISQED 2003. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE).