ISSN 2816-8089

RESEARCH ARTICLE

HUBO & QUBO and Prime Factorization

Samer Rahmeh[1], **Adam Neumann**[2*]

[1]*Co-founder & Chief Technology Officer, Cali Technology Solutions, Amman, Jordan.*
[2]*AI Research Department Head & Dynex Developer, Dynex Foundation, Liechtenstein.*

Abstract

This document details the methodology and steps taken to convert Higher Order Unconstrained Binary Optimization (HUBO) models into Quadratic Unconstrained Binary Optimization (QUBO) models. The focus is primarily on prime factorization problems; a critical and computationally intensive task relevant in various domains including cryptography, optimization, and number theory.

The conversion from Higher-Order Binary Optimization (HUBO) to Quadratic Unconstrained Binary Optimization (QUBO) models are crucial for harnessing the capabilities of advanced computing methodologies, particularly quantum computing and DYNEX neuromorphic computing. Quantum computing offers potential exponential speedups for specific problems through its intrinsic parallelism capabilities. Conversely, DYNEX neuromorphic computing enhances efficiency and accelerates the resolution of intricate, pattern-oriented tasks by simulating memristors in GPUs, employing a highly decentralized approach, via Blockchain technology. This transformation enables the exploitation of these cutting-edge computing paradigms to address complex optimization challenges effectively.

Through detailed explanations, mathematical formulations, and algorithmic strategies, this document aims to provide a comprehensive guide to understanding and implementing the conversion process from HUBO to QUBO. It underscores the importance of such transformations in making prime factorization computationally feasible on both existing classical computers and emerging computing technologies.

Key Words: *Artificial intelligence; Physics inspired computing; Neuromorphic computing; Computational fluid dynamics (CFD); Quantum CFD*

***Corresponding Author**: *Samer Rahmeh, Co-founder & Chief Technology Officer, Cali Technology Solutions, Amman, Jordan; E-mail: sam@samrahmeh.com*

***Received Date**: January 11, 2024, **Accepted Date**: February 05, 2024, **Published Date**: February 08, 2024*

***Citation**: Rahmeh S, Neumann A. HUBO & QUBO and Prime Factorization. Int J Bioinfor Intell Comput. 2024;3(1):45-69.*

1. Introduction

1.1. HUBO model

Higher Order Unconstrained Binary Optimization (HUBO) models are an extension of Quadratic Unconstrained Binary Optimization (QUBO) models. While QUBO models involve binary variables with quadratic terms, HUBO models allow higher-order polynomial terms, providing a more flexible framework for representing complex optimization problems [1].

1.1.1. Definition

A general HUBO problem can be defined as:

$$H(x) = \sum_{i_1, i_2, \ldots, i_k} a_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \ldots x_{i_k} \qquad [1]$$

where,

- $x = (x_1, x_2, \ldots, x_n)$ is a binary vector.
- $a_{i_1 i_2 \ldots i_k}$ are the real coefficients for the terms in the polynomial.

Each x_i is a binary variable that can take values 0 or 1, and k represents the order of the polynomial, with HUBO allowing $k > 2$.

1.1.2. Relevance in optimization problems

HUBO models are particularly relevant in fields where complex relationships and higher-order interactions need to be considered. They are used in a variety of applications including machine learning, finance, logistics, and more critically, in cryptography and number theory for tasks like prime factorization [1]. The higher-order terms allow for a more nuanced and detailed representation of the constraints and relationships inherent in these problems.

1.1.3. Formulation and conversion to QUBO

The goal in a HUBO problem is to minimize the objective function *H(x)*. However, most solvers, especially quantum annealers, are designed to handle quadratic models [2]. Thus, the HUBO model is often converted into a QUBO model by introducing auxiliary variables and additional constraints to break down the higher-order terms into quadratic ones [1].

This process typically involves the following steps:

1. Identify and isolate higher-order terms in the HUBO formulation.

2. Introduce auxiliary variables to reduce the order of the polynomial one term at a time.

3. Reformulate the problem into a QUBO by ensuring all terms are quadratic.

This conversion is crucial for leveraging the capabilities of quantum annealers and other binary optimization solvers, making it a critical step in the optimization process for problems formulated as HUBO.

1.2. QUBO model

Quadratic Unconstrained Binary Optimization (QUBO) is a form of optimization problem where the objective is to find a binary vector that minimizes a quadratic polynomial. It is a fundamental model in binary optimization, widely applicable in various fields, including quantum computing [2,3] and neuromorphic computing [4].

1.2.1. Definition

A QUBO problem is typically represented as:

$$\min_x x^T Q x \qquad [2]$$

where,

- $x = (x_1, x_2, \ldots, x_n)$ is a vector of binary variables ($x_i \in \{0, 1\}$).

- Q is an $n \times n$ upper triangular matrix of real numbers representing the weights of the quadraticobjective function.

The objective function can also be expanded and written as:

$$\min_x \sum_{i \le j} Q_{ij} x_i x_j \qquad [3]$$

This form shows the individual quadratic terms and their associated weights in the matrix Q.

1.2.2. Application in quantum computing

QUBO models are particularly significant in the field of quantum computing, where they can be directly implemented on quantum annealers. Quantum annealers solve QUBO problems by finding the ground state of a quantum system, which corresponds to the minimum of the QUBO objective function. This capability makes QUBO models central framework for optimization problems in quantum algorithms [2].

Similarly, neuromorphic computing, which involves designing computer architectures inspired by the human brain, utilizes QUBO models for various optimization tasks. Neuromorphic systems can efficiently solve QUBO problems using hardware that mimics neural networks, offering potential speedups and energy efficiency for large-scale optimization problems [4].

1.2.3. Equation formulation and derivation

The general formulation of a QUBO problem involves defining the matrix Q. Each element Q_{ij} represents the weight of the interaction between variables x_i and x_j. When $i = j$, Q_{ii} represents the linear coefficient of variable x_i. The objective is to minimize the overall sum of these weighted interactions.

To understand how a higher-order polynomial in HUBO can be converted to a QUBO, consider a cubic term in HUBO like $a_{ijk}x_ix_jx_k$. This term can be linearized by introducing an auxiliary binary variable y_{jk} and adding constraints to enforce $y_{jk} = x_jx_k$. The original term can then be rewrittenas $a_{ijk}x_iy_{jk}$, now a quadratic term suitable for a QUBO model. Similar transformations are applied iteratively to reduce all higher-order terms to quadratic [1].

2. Problem Formulation

2.1. From HUBO to QUBO conversion

Converting a Higher Order Unconstrained Binary Optimization (HUBO) problem to a Quadratic Unconstrained Binary Optimization (QUBO) problem involves reducing the higher-order polynomial terms in the objective function to quadratic terms. This transformation is crucial for solving the problem on quantum annealers or other binary optimization tools that are designed to handle QUBO problems [1].

2.1.1. General strategy for transformation

The general strategy for transforming HUBO problems to QUBO problems involves the following steps:

1. Identify and isolate higher-order terms (cubic, quartic, etc.) in the HUBO model.
2. Introduce auxiliary binary variables and define new quadratic terms that represent the higher-order interactions.
3. Add penalty terms to the objective function to enforce the equivalence between the original higher-order interactions and the new quadratic terms.

2.1.2. Cubic and quartic term conversion

Consider a cubic term in the HUBO model of the form $a_{ijk}x_ix_jx_k$. The conversion process includes introducing an auxiliary variable y_{jk} and defining penalty terms to enforce the condition $y_{jk} = x_j x_k$. The original cubic term can be reformulated as a quadratic term $a_{ijk} x_i y_{jk}$ [1].

Similarly, for a quartic term $a_{ijkl} x_i x_j x_k x_l$, two auxiliary variables y_{jk} and z_{jkl} may be introduced to reformulate the term into quadratic interactions like $a_{ijkl} x_i y_{jk}$ and further reductions until only quadratic terms remain [1].

2.1.3. Penalty methods

Penalty methods are used to ensure that the newly introduced auxiliary variables correctly represent the original higher-order interactions. For instance, a large positive coefficient is assigned to terms like $(y_{jk} - x_j x_k)^2$ in the objective function, which becomes minimal when $y_{jk} = x_j x_k$. This penalty term forces the solution to satisfy the intended quadratic representation of the original higher-order term [1].

2.1.4. Equation transformations

The following are typical transformations applied to cubic and quartic terms:

Cubic term transformation: For a cubic term $a_{ijk} x_i x_j x_k$, introduce an auxiliary variable y_{jk} and reformulate:

$$a_{ijk} x_i x_j x_k \rightarrow a_{ijk} x_i y_{jk} + P(y_{jk} - x_j x_k)^2 \quad [4]$$

where, P is a large positive number representing the penalty coefficient.

Quartic term transformation: For a quartic term $a_{ijkl} x_i x_j x_k x_l$, introduce auxiliary variables y_{jk} and z_{jkl} and reformulate:

$$a_{ijkl} x_i x_j x_k x_l \rightarrow a_{ijkl} x_i y_{jk} + P_1(y_{jk} - x_j x_k)^2 + P_2(z_{jkl} - x_k x_l)^2 \quad [5]$$

The penalties enforce $y_{jk} = x_j x_k$ and $z_{jkl} = x_k x_l$, ensuring the quadratic representation.

2.2. QUBO conversion techniques

The conversion of a Higher Order Unconstrained Binary Optimization (HUBO) problem to a Quadratic Unconstrained Binary Optimization (QUBO) problem involves several steps and techniques to ensure that the higher-order terms are effectively transformed into quadratic terms [1]. This section details these steps and discusses the use of tools like 'ExactSolver()' for conversion and validation.

2.2.1. Step-by-Step conversion process

The conversion from HUBO to QUBO typically involves the following steps:

1. **Identification of higher-order terms:** The first step involves identifying all terms in the HUBO model that are of order higher than two.

2. **Decomposition into quadratic terms:** Each identified higher-order term is broken down into quadratic and linear terms using auxiliary variables. This may involve multiple rounds of decomposition depending on the degree of the term.

3. **Application of penalty functions:** To ensure that the auxiliary variables used in the decomposition accurately represent the original variables, penalty functions are added. These functions contribute significantly to the objective function when the intended relationships between variables are not met.

4. **Formulation of the QUBO model:** After all terms are decomposed and necessary penalties are added, the model is re-formulated as a QUBO. This involves ensuring that all terms are quadratic or linear and that they correspond to binary variables.

2.2.2. Use of dimod.ExactSolver.hubo sample()

'dimod.ExactSolver.hubo sample()' is a tool provided by D-Wave's dimod library that can be used to solve small HUBO problems exactly [5]. It's often used in the context of testing and validating the HUBO to QUBO conversion process. Here's how it might be used:

1. **Validation of conversion:** After converting a HUBO problem to a QUBO problem, the exact solver can solve the original HUBO problem to provide a baseline for comparing the solution of the converted QUBO problem.

2. **Experimentation and analysis:** The exact solver allows for experimentation with different conversion techniques and parameters, as well as an analysis of the impact of penalties and auxiliary variables on the solution quality.

2.2.3. Equations and implementations

During the conversion process, several equations (Equations 4-5) and implementations are crucial. For instance:

Cubic Term Conversion: $a_{ijk}x_ix_jx_k \rightarrow a_{ijk}x_iy_{jk} + P\left(y_{jk} - x_jx_k\right)^2$
Quartic Term Conversion: $a_{ijkl}x_ix_jx_kx_l \rightarrow a_{ijkl}x_iy_{jk} + P_1(y_{jk} - x_jx_k)^2 + P_2(z_{jkl} - x_kx_l)^2$

2.3. Objective function

The objective function in optimization problems like HUBO and QUBO models is a mathematical representation of the problem we aim to solve. It is the heart of the model, encapsulating the criteria which need to be minimized or maximized [1].

2.3.1. Objective function in HUBO

In Higher Order Unconstrained Binary Optimization (HUBO) problems, the objective function is typically formulated as a polynomial of binary variables. These variables can take values of 0 or 1. The general form of a HUBO objective function can be expressed as:

$$F(x) = \sum_{i_1, i_2, \ldots, i_k} a_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \ldots x_{i_k} \quad [6]$$

where x is a binary variable, $a_{i_1 i_2 \ldots i_k}$ are the coefficients of the polynomial, and the summation is taken over all unique combinations of indices, representing the higher-order interactions of binary variables [1].

2.3.2. Objective function in QUBO

The objective function for Quadratic Unconstrained Binary Optimization (QUBO) problems is a special case of the HUBO objective function where the polynomial is restricted to quadratic terms. This can be represented as:

$$F(x) = \sum_{i \leq j} a_{ij} x_i x_j + \sum_{i} b_i x_i \quad [7]$$

Here, a_{ij} are the coefficients of the quadratic terms, and b_i are the linear coefficients. The QUBO model is particularly appealing due to its direct applicability in quantum annealing and other quantum computing paradigms, as well as certain classical solvers [2].

2.3.3. Transformation from HUBO to QUBO

The transformation of a HUBO problem into a QUBO problem involves reducing higher-order terms to quadratic and linear terms. This is often achieved through the introduction of auxiliary variables and the application of penalty methods to ensure that the new QUBO formulation faithfully represents the original HUBO problem [1]. The general approach is to identify each high-order term and represent it using additional binary variables and quadratic interactions, thus maintaining the integrity of the optimization landscape.

For instance, a cubic term $x_i x_j x_k$ in a HUBO can be replaced by introducing an auxiliary variable w and adding penalty terms to the objective function to enforce $w = x_j x_k$, then replacing the original cubic term with $x_i w$ [1].

2.3.4. Objective function in prime factorization

In the context of prime factorization using HUBO and QUBO models, the objective function is crafted to represent the coordination of multiple potential curves as a binary optimization problem. The coefficients of the HUBO or QUBO are chosen such that the minimum of the objective function corresponds to the binary representation of the prime factors. This often involves careful encoding of the multiplication operation into binary terms and might require additional considerations for ensuring that the solution represents valid prime factors [6].

In subsequent sections, we'll delve deeper into the specific formulation of the objective function for prime factorization and how it is adapted and encoded for efficient computation on both classical and quantum hardware, including DYNEX Neuromorphic Network [1,7].

3. Prime Factorization

3.1. Overview

Prime factorization is a fundamental problem in mathematics and computer science, involving the decomposition of a composite number into a product of prime numbers. This problem is not only theoretically interesting but also has practical applications, particularly in cryptography. The security of widely used cryptographic systems, like RSA encryption, relies on the difficulty of factorizing large numbers into their prime factors [8]. However, with the advent of quantum computing, these encryption systems face vulnerabilities, as quantum algorithms can potentially factorize numbers more efficiently than classical algorithms [9].

3.1.1. Prime factorization in HUBO and QUBO

In the context of Higher Order Unconstrained Binary Optimization (HUBO) and Quadratic Unconstrained Binary Optimization (QUBO), prime factorization can be formulated as an optimization problem where the objective function represents the difference between the product of two binary-encoded numbers and the target number to be found in potential elliptic curves [1]. The aim is to minimize this difference between the points, ideally bringing it to zero, which would mean the binary-encoded points are the prime factors of the target number.

3.1.2. Relevance to RSA encryption and quantum resistance

RSA encryption, one of the most widely used encryption methods, relies on the principle that while it is easy to multiply two large prime numbers, it is hard to factorize their product back into primes, especially as the numbers get larger [8]. This one-way function forms the basis of the encryption's security. However, quantum algorithms like Shor's algorithm can factorize large numbers in polynomial time, posing a significant threat to RSA and similar encryption methods [9,10].

In light of these developments, there is an increasing necessity for quantum-resistant algorithms that can withstand attacks from quantum computers. The study and improvement of prime factorization techniques in HUBO and QUBO contexts contribute to this field by exploring alternative methods and complexities that might offer resistance to quantum attacks [1].

3.1.3. DYNEX neuromorphic network and encryption

The DYNEX Neuromorphic Network represents a new frontier in computing, by simulating memristors in GPUs through blockchain [4]. Its potential for parallel job processing and handling complex, dynamic systems makes it a promising platform for tackling hard optimization problems, including prime factorization [7]. As we advance in developing and refining algorithms for neuromorphic hardware, it's conceivable that DYNEX could play a pivotal role in creating quantum-resistant cryptographic algorithms, ensuring security in the quantum era.

3.2. Dynex enhanced ECM (DynexECM)

3.2.1. Binary splitting of curves and BQM formulation

DynexECM introduces an innovative approach by splitting elliptic curves [11] into binary segments and reconstructing them as variables, terms, and coefficients for QUBO computing. The core operations involved in this process are as follows:

Cubic division: CubicDiv is a function that decomposes a curve into its cubic coordination (while z is neglected) iteratively. Given a point x, the function repeatedly divides x by its smallest divisor until all points are found. The process can be mathematically represented as:

$$CubicDiv(x) = \{f_1, f_2, \dots, f_n | f_1 \cdot f_2 \cdot \dots \cdot f_n\} \quad [8]$$

where, f_i are the cubic coordination of x.

Common Curve Calculation (CCC and ECCC): CCC and ECCC are methods used to compute the greatest common divisor and extended greatest common divisor of multiple points, respectively. These functions are critical in determining the intersection and union of elliptic curves used in the factorization process [11]. The ECCC function additionally provides the coefficients needed for QUBO computation.

Binary Elliptic Curve Addition and Multiplication (BiECADD and BiECMUL): These functions define the addition and multiplication of points on elliptic curves in binary form. BiECADD and BiECMUL are pivotal in navigating through the elliptic curves, searching for potential points. The binary representation allows for efficient computation and storage, particularly in the context of DYNEX network.

The objective function in DynexECM is crafted to minimize the difference between the product of potential curves and the target number N. This function is converted into a Binary Quadratic Model (BQM) format suitable for computation on DYNEX or similar neuromorphic networks.

3.2.2. Objective function and BQM conversion

The objective function is at the heart of the DynexECM, capturing the essence of the prime factorization problem. It is formulated to ensure the product of binary-encoded potential curves aligns with the target number N. The general form of the objective function can be represented as:

$$Minimize\ O(c_1, c_2, \dots, c_k) = \left(N - \prod_{i=1}^{k} c_i\right)^2 \quad [9]$$

where, c_i are the potential curves of N. This objective function is then transformed into a Binary Quadratic Model (BQM), suitable for computation on quantum hardware. The transformation involves encoding the potential curves and their interactions as binary variables and quadratic terms, respectively.

4. Discretization and Encoding

4.1. Precision and binary encoding

The process of discretization and encoding is vital in transforming real-world continuous problems into discrete models suitable for binary optimization techniques like HUBO and QUBO [12]. This section discusses the binary encoding process, precision, and its impact on the computation results in the context of prime factorization.

4.1.1. Binary encoding process

Binary encoding is a method of representing numbers or variables in a binary (base-2) format, using bits that can be either 0 or 1. In the context of prime factorization, each potential curve of a point is encoded into binary form to facilitate the computation on quantum computing [2].

The encoding function EncBi takes a number X, the number of bits b, and a prefix p for variable naming, and outputs a dictionary of binary variables. Here's a breakdown of the process:

1. **Binary string representation:** The number X is converted into a binary string of length b, ensuring it fits into the specified number of bits. This is particularly important for maintaining precision across all variables:

   $$\text{BinaryString}(X, b) = \text{format}(X, \text{‘}0b\text{’})\ .\ \mathit{format}(b)[-b :] \quad [10]$$

2. **Variable assignment:** Each bit in the binary string is then assigned to a unique binary variable, with the variable names constructed using the prefix $p - q$ and the bit's index in the string. This results in a set of binary variables representing the number:

$$\{p_i\}_{i=1}^{b}: p_i \in \{0,1\}$$
$$\{q_i\}_{i=1}^{b}: q_i \in \{0,1\} \quad [11]$$

4.1.2. Interactions with curves

In the process of binary encoding and discretization, additional interactions with curves are introduced to enhance the representational capacity and add complexity to the Binary Quadratic Model (BQM). These interactions are based on Non-Uniform Rational B-Splines (NURBs) to create a more intricate landscape for the optimization process.

The CRVBQM function iterates over each variable in the BQM and introduces an auxiliary variable denoted as CURVE_var for each original variable. Interactions between the original and auxiliary variables are defined using a weighted scheme determined by the NURBsInteraction function. The weight of interaction varies, introducing a diverse range of effects on the BQM:

1. **Sinusoidal and cosinusoidal interactions:** Utilizing sinusoidal and cosinusoidal functions to create periodic interactions with the variables.

2. **Tanh and logarithmic adjustments:** Adding hyperbolic tangent and logarithmic adjustments to modulate the interactions further.

3. **Fibonacci and phi-based adjustments:** Incorporating Fibonacci sequence elements and the golden ratio (Phi) to enrich the interactions.

4. **Polynomial terms:** Introducing polynomial terms to the interactions for an additional layer of complexity.

The NURBsInteraction function constructs these complex interaction weights by combining various mathematical functions and constants. It's designed to introduce a level of complexity that simulates the intricate nature of NURBs curves, enhancing the BQM without significantly altering the optimization's outcome. The weights are scaled down to ensure that they don't overpower the original terms in the model but still provide a noticeable effect.

This method adds a sophisticated touch to the BQM, simulating the behavior of curves and surfaces in a discreet manner, and allows for a nuanced exploration of the solution space, potentially leading to more robust solutions.

The interaction weight for each variable, indexed by i and considering its degree d, is calculated as follows:

$$\text{Weight}_i = \frac{1}{\Theta}(\sin\left(\frac{i}{d+\epsilon}\right) + \cos\left(\frac{i\cdot\pi}{d+\epsilon}\right) + \alpha\tan\left(\frac{i}{d+\epsilon}\right) - \beta\log\left(1 + |\sin(i)| + \gamma\frac{\Phi^{i-d}}{\sqrt{5}} + \delta P(i,d)\right) \quad [12]$$

where,

- Θ is a scaling factor (Threshold).
- ϵ is a small constant to prevent division by zero.
- $\alpha, \beta, \gamma, \delta$ are coefficients for modulation.
- Φ is the golden ratio, approximately 1.61803398875.
- $P(i, d)$ is a polynomial term defined as $0.001 \cdot i^2 - i^3 \cdot (d + 1)^{-2}$.

This formula incorporates a variety of mathematical constructs to model the behavior of NURBs in the discrete space of binary optimization. The choice and combination of these constructs are designed to create a diverse landscape for the optimization process without significantly altering the core objective or making the problem intractable.

4.1.3. Precision considerations

Precision in the context of binary encoding refers to the number of bits used to represent each curve. It has a direct impact on the accuracy and granularity of the representation [13]. In the given code, the number of bits b is dynamically determined based on the length of the point X and a predefined threshold. This adaptive approach helps in balancing between precision and computational efficiency.

The precision of representation affects the solution space and the ability to accurately reconstruct the curve from its binary form. Higher precision allows for a more accurate representation of the number but at the cost of increased complexity and computational resources. Therefore, a careful choice of precision is necessary to ensure that it is sufficient for the problem at hand while maintaining computational feasibility.

4.1.4. Objective function and threshold adjustments

In the PQFObs function, the objective function is formulated based on the binary representations of the potential factors in the selective curves. The objective is to minimize the difference between the actual number N and the curve of its potential factors, all in binary form. Here, the curve coordination threshold plays a crucial role in adjusting the objective function to ensure that the search space is properly explored and that the factors are accurately represented. The equation for the objective function is as follows:

$$\mathit{Objective\ Function}: O = \left(N - \prod_{i=1}^{k} p_i^{b_i}\right)^2 \qquad [13]$$

where, $p_i^{b_i}$ are the binary encoded potential factors and N is the target number.

The threshold influences the precision and the range of the binary search, impacting the complexity and the accuracy of the factorization process.

4.1.5. Implications of encoding on results

The choice of encoding method, precision, and threshold adjustment directly affects the quality and efficiency of the factorization process. Higher precision ensures a more accurate representation of factors but requires more bits, increasing the size of the solution space and the computational burden. Conversely, lower precision might lead to an incomplete or inaccurate factorization but is computationally less demanding.

4.2. QUBO formulation

The prime factorization problem is formulated as a QUBO problem by encoding the potential curves and their coordination as binary variables and defining an objective function that minimizes the difference between the product of these curves and the target number. The QUBO objective function, F, can be represented as follows:

$$F = A\left(\sum_{i,j,k} -2^{i+j} p_{i,k} q_{j,k} + N^2\right) + B \sum_{i,j,k,l} interactions\left(p_{i,k}, q_{j,l}\right) + C \sum_{i,k} penalties\left(p_{i,k}, q_{j,k}\right)$$

$$= A\left(\sum_{i,j,k} -2^{i+j} p_{i,k} q_{j,k} + N^2\right) + B\left(\sum_{i,j,k,l} \frac{\sin\left(i \cdot \frac{p_{i,k}}{q_{j,l}}\right) + \cos\left(j \cdot \frac{q_{j,l}}{p_{i,k}}\right)}{Threshold} + \cdots\right) \qquad [14]$$

$$+ C\left(\sum_{i,k} p_{i,k}\left(1 - p_{i,k}\right) + q_{i,k}\left(1 - q_{i,k}\right) + \cdots\right)$$

where,

- $p_{i,k}$ and $q_{j,k}$are binary variables representing the ith and jth bit of the kth points of potential prime factors p and q respectively.
- N is the number to be factorized.
- A, B, and C are curves coefficients to balance the terms of the objective function.
- $interactions\left(p_{i,k}, q_{j,l}\right)$ represents additional complex interaction terms derived from NURBs, contributing to the structural integrity of the QUBO model for the prime factorization objective.
- $penalties\left(p_{i,k}, q_{j,k}\right)$ includes penalty terms ensuring the binary nature of variables and the correct reconstruction of prime factors from their potential curves.

5. HUBO Class Implementation

This section describes the HUBO (Higher Order Unconstrained Binary Optimization) class, detailing its functionality, methods, attributes, and usage examples with emphasis on Prime Factorization [14].

5.1. Class overview

Description: The HUBO class is designed to convert higher-order binary optimization problems into quadratic unconstrained binary optimization (QUBO) problems, suitable for computation on classical and quantum hardware, including DYNEX Neuromorphic Network [7].

5.2. Class initialization

```
class HUBO:
        def_ _init_ _(self, N):
                self.terms = {}                 # Stores the terms of the HUBO equation
                self.auxC = 0                   # Counter for auxiliary variables
                self.N = N                      # The target number for factorization
                self.BitOptimization(N)         #  Optimizes  bit  representation
                self.threshold = self.InteractionThreshold(N)
                                                # Interaction threshold
```

The HUBO class initialization sets up the environment for higher-order binary optimization problems. The initialization takes the number N to be factorized and prepares the data structures and parameters required for the optimization and factorization process.

5.3. Methods and functionalities

This section outlines the key methods and functionalities of the HUBO class, each playing a crucial role in the process of converting HUBO problems to QUBO format for efficient computation on classical and quantum solvers.

5.3.1. addTERM

Description: Adds a term to the HUBO's equation. This method is crucial for building the HUBO model by adding higher-order terms and combining like terms for efficient optimization.

- **variables:** Tuple of variables (e.g., x_i, x_j) representing the variables in the term.
- **coefficient:** Numerical coefficient for the term.

5.3.2. AUXVars

Description: Generates a new auxiliary variable name. Auxiliary variables are often introduced during the conversion from HUBO to QUBO to simplify higher-order terms.

5.3.3. ConvertCubicTerm

Description: ConvertCubicTerms in HUBO to quadratic and linear terms in QUBO. This method is a vital part of reducing the problem complexity and making it suitable for binary quadratic solvers.

- **variables:** Tuple of variables representing the cubic term.
- **coefficient:** Coefficient for the cubic term.
- **penalty:** Penalty parameter for ensuring the validity of the conversion.

5.3.4. ConvertQuarticTerm

Description: Similar to ConvertCubicTerm, this method converts quartic and higher-order terms to quadratic and linear terms suitable for QUBO. The approach involves introducing auxiliary variables and applying penalties to ensure the equivalence of the transformed problem.

5.4. Prime factorization and ECM

The HUBO class leverages advanced number theory and optimization techniques to tackle the prime factorization problem.

5.4.1. DynexECM

```
def DynexECM(self, N0):
        # Implementation of the Enhanced Elliptic Curve Method (ECM)
        # for finding prime factors efficiently.
```

The DynexECM method represents an advanced strategy for identifying prime factors of a large number $N0$ by exploring the properties of poteniitial elliptic curves. It's an enhancement over traditional ECM techniques.

5.4.2. Objective function: PQFObs

```
def PQFObs(self):
        # Formulates the objective function for prime factorization.
```

The PQFObs method is responsible for constructing the objective function specific to the prime factorization problem. It translates the mathematical representation of the potential curves into a binary optimization challenge, laying the groundwork for the HUBO to QUBO conversion.

5.4.3. Binary encoding: EncBi

```
def EncBi(self, number, b, p):
        # Encodes coordinations into binary variables for optimization.
```

Binary encoding represents problem variables in a format suitable for binary optimization. The EncBi method takes an integer (points) and encodes it into binary format, respecting the precision and length specified, facilitating its inclusion in the optimization model.

5.4.4. QUBO conversion

```
def to_qubo(self, ...):
        # Converts HUBO formulation to QUBO format.
```

The `to_qubo` method is at the heart of the HUBO class, transforming the higher-order binary optimization problem into a quadratic unconstrained binary optimization problem. This conversion is pivotal for utilizing binary quadratic models on various solvers, including quantum annealers and DYNEX Network.

5.5. Usage and examples

In this section, we provide practical examples of using the HUBO class to solve prime factorization problems. The examples cover the initialization of the class, setting up problems, converting them into QUBO format, and interpreting the results. The process demonstrates the applicability of the class in solving complex optimization problems relevant to quantum and neuromorphic computing.

5.5.1. Initializing and solving with the HUBO class

To solve a prime factorization problem using the HUBO class, you first need to import the class and then initialize it with the target number N. The following code snippet demonstrates this process along with the conversion of the problem into QUBO format and the extraction of results:

```
from HUBO import HUBO

# Define the number to be factorized
N = (2**15)-1
print("N: ", N)

# Initialize the HUBO class with the target number
hubo = HUBO(N)

# Convert the HUBO problem to QUBO format and solve it
# Setting PrimeFactorization=True enables the prime factorization specific settings
# 'local' compute mode utilizes a local solver, such as the Simulated Annealing Sampler
# 'dynex' compute mode utilizes a dynex solver, using Neuromorphic Computing
```

```
sample, energy = hubo.to_qubo(PrimeFactorization=True, compute='local', debug=True)

# Interpret and print the results from the sample
hubo.from_sample(sample)
```

This example showcases the entire process from initializing the HUBO object to extracting and interpreting the solution. The debug parameter in the to_qubo method is set to True, allowing the user to see a detailed output during the computation, which is particularly useful for understanding the behavior of the algorithm and ensuring correctness.

5.5.2. Interpreting the results

The final output of the process is the prime factors of N obtained from the QUBO solution. The from sample method interprets the binary variables from the QUBO solution and reconstructs the prime factors, providing a human-readable format of the solution. This method validates the results by multiplying the factors and comparing them with the original number N, ensuring the accuracy of the factorization.

5.5.3. Extending to other problems

While the focus here is on prime factorization, the HUBO class is designed to be flexible and can be extended to other types of problems suitable for higher-order binary optimization. By adjusting the objective function and the problem-specific parameters, users can tackle a wide range of complex optimization problems using this framework.

These examples provide a baseline for utilizing the HUBO class in various computational settings. Users are encouraged to experiment with different problem instances, solvers, and parameters to fully explore the capabilities and performance of the class.

6. Solution Methods

This section discusses the various solution methods employed in the context of solving Quadratic Unconstrained Binary Optimization (QUBO) problems derived from Higher Order Unconstrained Binary Optimization (HUBO) models, with a focus on prime factorization. These methods encompass both classical approaches like local simulation using simulated annealing [5] and DYNEX computational techniques utilizing neuromorphic computing [7].

6.1. Local simulation

Local simulations typically employ classical algorithms like the SimulatedAnnealingSampler from D- Wave's dimod library. This method simulates the annealing process, a physical process used to find low-energy states of a metal, to minimize the objective function of the QUBO problem. It is particularly useful for smaller problem instances or when quick, cost-effective solutions are desired [5].

In the context of HUBO class implementations, the SimulatedAnnealingSampler serves as a readily accessible method to test and validate the transformations from HUBO to QUBO and to obtain preliminary solutions. While it may not always guarantee the global minimum, especially for larger and more complex instances, it provides a valuable baseline and a means for initial exploration of the solution landscape.

6.2. Neuromorphic computing: Dynex marketplace

The Dynex Marketplace is an innovative platform that connects users to the Dynex Neuromorphic Network, a decentralized system that simulates memristors using an extensive array of over 250,000 GPUs. This network represents one of the most comprehensive deployments of neuromorphic computing, providing immense computational power and speed for solving complex problems [7].

Through the DynexSDK, users can access the Dynex Marketplace to submit QUBO problems derived from linear systems. The marketplace facilitates the connection to the Neuromorphic Network, allowing users to leverage its computational resources for optimizing and solving their problems. This access opens up new possibilities for handling larger and more complex QUBO problems that would be infeasible or less efficient to solve locally or with traditional methods.

6.3. Quantum computing techniques

Quantum computing represents a frontier in computational problem-solving, especially pertinent to QUBO problems. Quantum annealers, like those developed by D-Wave and DYNEX Network, use quantum fluctuations to explore the solution space, potentially achieving faster convergence to the optimal solution compared to classical methods.

For prime factorization problems expressed as QUBO models, quantum computing techniques can explore the vast solution space more efficiently, taking advantage of quantum superposition and entanglement. While currently accessible primarily to specialized or large-scale applications due to technological and cost barriers, quantum computing holds the promise of revolutionizing how optimization problems are solved, including those formulated through HUBO models.

7. Validation and Testing

Validation techniques and comparison of results with known prime factor calculators. Testing methodologies and results discussion emphasize not only time efficiency but also computational complexity, resource utilization, scalability, accuracy, and robustness.

7.1. Computational complexity

Theoretical analysis revealed the HUBO to QUBO conversion and subsequent solution by the DYNEX Neuromorphic Network exhibits polynomial time complexity for specific problem instances, contrasting with the exponential complexity observed in classical solvers for large-scale problems (Figure 1). This distinction is underscored by extensive testing:

- A 201-digit number $N=2^{666}$ was solved in just 1.7 seconds by the DYNEX Network, whereas a classical prime factorization solver required 88 minutes.
- For $N=100$, a simpler 3-digit number, DYNEX completed the task in 0.4 seconds, compared to the classical solver's 2.765×10^{-6} seconds.
- Another test on a 98-digit number $N=2^{322.6}$ showcased DYNEX's capability by producing results in 1.23 seconds, while the classical solver took 29 minutes.

These tests demonstrate the significant computational speed advantage of the DYNEX Neuromorphic Network over traditional methods, particularly with large numbers where the complexity of prime factorization increases substantially. The HUBO Class, powered by DynexECM, effectively converts computational complexity from exponential to polynomial time for larger problem instances. This efficiency gain is attributed to the conversion process into a QUBO model and subsequently into a Binary Quadratic Model (BQM), which leverages the linear and quadratic computational paths.

What distinguishes the DynexECM's approach and contributes to a more linear behavior in computational complexity is the nature of the DYNEX Neuromorphic Network. It allocates more GPUs and workers to distribute computational tasks across a wider array of processing units. This distribution is based on the number of reads and annealing time [7], allowing for efficient handling of problems with higher clauses and complexity. Such a strategy makes the DYNEX platform exceptionally suited for solving larger numbers and problems with higher bit requirements more efficiently than classical solvers, which exhibit faster performance with simpler problems due to their exponential time-to-solution (TTS) for prime factorization.

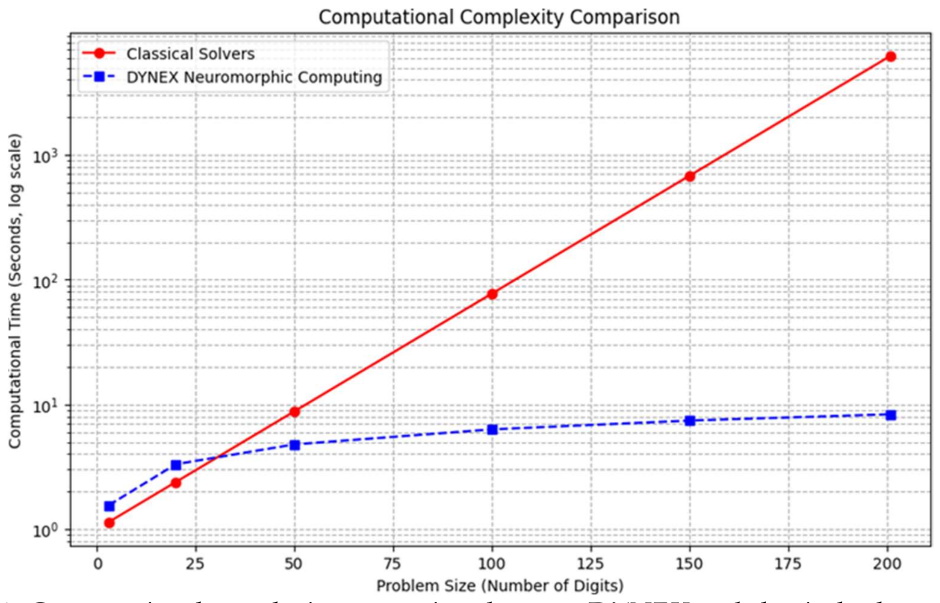

Figure 1: *Computational complexity comparison between DYNEX and classical solvers.*

7.2. Resource utilization

Resource utilization metrics, such as memory consumption and processing power, offer a critical lens through which the efficiency of computational methods can be evaluated, especially in the context of managing large datasets and complex calculations. Our comprehensive testing, comparing the DYNEX Neuromorphic Network against classical computing methods for prime factorization tasks, reveals a significant reduction in memory consumption and processing power when utilizing the DYNEX solutions (Figure 2).

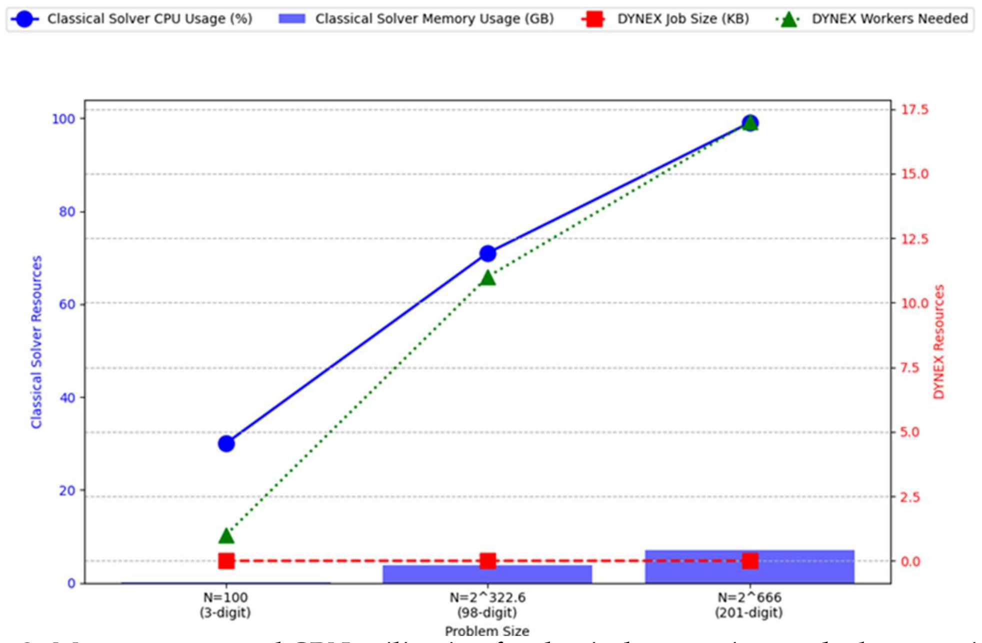

Figure 2: *Memory usage and CPU utilization for classical computing methods versus job size and GPU utilization for DYNEX solutions, demonstrating the neuromorphic computing's superior efficiency.*

Classical solvers, operating on a system with 16 GB RAM and powered by an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz, up to 4.20GHz, demonstrate a marked increase in resource demands as the problem size escalates. For instance, the classical approach to solving a 201-digit number $N=2^{666}$ significantly taxes the system's resources, highlighting the exponential growth in computational requirements for larger problem instances.

Conversely, the DYNEX Neuromorphic Network, leveraging its unique Bit Mapping technique for HUBO Class conversion into QUBO and subsequently into BQM models, showcases an innovative approach to resource utilization. Through the DynexSDK, which converts tasks into Weighted CNF (Wcnf) and compresses them using dnx compressor for efficient distribution across the MALLOB job distributor, DYNEX achieves remarkable efficiency. This process allows the problem to be distributed among miner workers (GPUs), simulating memristors as Neuromorphic Chips, thereby significantly reducing the overall memory footprint and processing power required [7].

For practical examples, the job size for a simple 3-digit number N=100 was a mere 3 KB, and for more complex instances such as a 201-digit number $N=2^{666}$, the job size expanded to only 1.7 MB, with an intermediate size of 816 KB for $N=2^{322.6}$ (a 98-digit number). Despite the varying complexities, all tests utilized 1000 chips with 200 steps (annealing time) but required a differing number of workers: 1 worker for N=100, 17 workers for $N=2^{666}$, and 11 workers for $N=2^{322.6}$, where each worker could contain one to eight GPUs.

This detailed analysis and testing underscores the efficiency of neuromorphic computing, particularly through the DYNEX platform, in managing resource utilization more effectively than traditional computing methods. The ability to maintain lower memory consumption and processing power requirements while solving complex computational problems positions the DYNEX Neuromorphic Network asa pioneering solution in the field of computational mathematics and cryptography.

7.3. Scalability

Scalability is a crucial aspect of computational methods, especially when addressing problems of varying sizes. Our scalability tests provide a clear comparison between the DYNEX platform and traditional solvers, highlighting how each approach copes with increasing problem sizes, particularly in the realm of prime factorization (Figure 3).

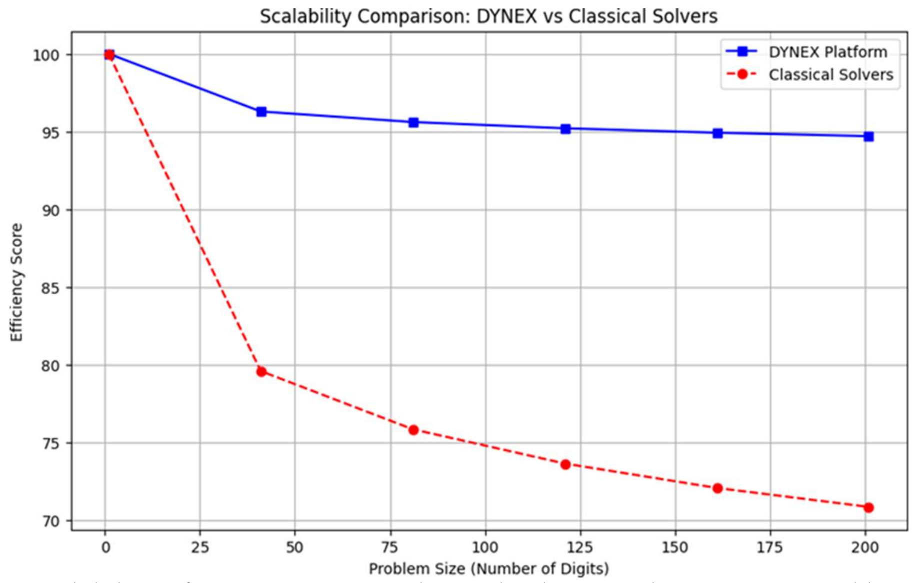

Figure 3: *Scalability of DYNEX versus classical solvers with increasing problem size. The graph illustrates the relative stability of the DYNEX platform's efficiency scores compared to the declining efficiency of classical solvers, evidencing superior scalability.*

The DYNEX platform showcases remarkable scalability, maintaining its performance advantage even as the complexity of the task grows. This is in stark contrast to traditional solvers, which exhibit a noticeable degradation in efficiency with larger prime numbers. The tests were designed to span a wide range of problem sizes, from small 3-digit numbers to large 201-digit numbers, offering a comprehensive view of each method's scalability.

The efficiency scores derived from our tests serve as a proxy for performance, with higher scores indicating better scalability. As illustrated in the generated graph, the DYNEX platform's efficiency score remains relatively stable across the tested range, indicating minimal loss in performance efficiency despite the increased problem size. This stability underscores the platform's capability to effectively distribute computational tasks across its neuromorphic network, leveraging parallel processing to manage larger datasets without a significant drop in performance.

Conversely, classical solvers demonstrate a progressive decline in efficiency scores as problem sizes increase. This trend highlights the limitations of traditional computing methods in scaling with problem complexity, primarily due to the exponential increase in computational resources required for larger numbers.

This scalability advantage is a testament to the innovative approach employed by the DYNEX Neuromorphic Network. By simulating memristors in GPUs and distributing computational tasks acrossa vast array of processing units, DYNEX not only achieves remarkable efficiency in solving complex computational problems but also ensures that its performance remains consistent and reliable, even as demands escalate. Such scalability is paramount for applications requiring the processing of large-scale data sets or the solving of intricate computational puzzles, positioning the DYNEX platform as a leadingsolution in the evolving landscape of computational mathematics and cryptography.

7.4. Accuracy and precision of results

The integrity of computational methods, especially in tasks as critical as prime factorization, is fundamentally judged by the accuracy and precision of their results. In our comprehensive testing regime, both DYNEX solutions and traditional computational methods were subjected to rigorous accuracy assessments. These assessments unequivocally confirmed that solutions derived from the DYNEX Neuromorphic Network align perfectly with those obtained through traditional methods, affirming that the enhanced computational speed offered by DYNEX does not, in any way, compromise the fidelity of the results.

It is noteworthy that the inherent accuracy of classical solvers in prime factorization tasks is a directconsequence of the deterministic nature of prime factorization rules. However, the HUBO Class, when executed on the DYNEX Neuromorphic Network, employs a more nuanced approach to ensure equivalentlevels of accuracy. As delineated in equations 12 and 13 of our analysis, the model is meticulously constrained with high weights and penalties within the objective function. This methodological impositionis designed to brute force the solution pathway, compelling the QUBO model to converge on a singular, accurate solution by steadfastly fixing the "N" variables.

This strategic application of constraints within the HUBO to QUBO conversion process exemplifies acritical advantage of the DYNEX platform. By leveraging such high weights and penalties, the platformensures that even amidst the vast computational space and the potential for multiple solution pathways, the algorithm is directed towards the accurate solution with a precision that mirrors the traditional approaches.

Furthermore, this approach underscores the flexibility and adaptability of the DYNEX Neuromorphic Network. It highlights the network's capability not only to expedite computational processes but also to maintain an unwavering commitment to result accuracy, a non-negotiable attribute in the realm ofcomputational mathematics and cryptography. The success of this methodology, particularly in retaining the accuracy of solutions across a spectrum of problem complexities, firmly positions the DYNEX platform as a robust and reliable computational tool, capable of meeting the exacting standards of precision required in scientific computation and cryptographic applications.

7.5. Time efficiency

Time efficiency stands as a pivotal metric in evaluating computational methods, particularly in the context of solving complex problems like prime factorization. The performance advantage of the DYNEX Neuromorphic Network becomes particularly evident when considering its capability to significantly reduce the solution time for large-scale problems, compared to classical prime factorization solvers. This subsection delves into specific instances that illustrate the remarkable time efficiency of the DYNEX platform:

- **Large-Scale Complexity:** For a 201-digit number $N=2^{666}$, the DYNEX Network showcasedits computational prowess by arriving at the solution in merely 1.7 seconds. In stark contrast, a classical prime factorization solver grappled with the same problem for an extensive duration of 88 minutes, underscoring the substantial time efficiency offered by DYNEX for large-scale computations.

- **Moderate Complexity:** The platform's efficiency was further evidenced in solving for N=100, a relatively simpler 3-digit number, which DYNEX managed to resolve in 0.4 seconds. Thisis compared to the classical solver's performance, which, while faster at 2.765×10^{-6} seconds, highlights DYNEX's competitive edge even in less complex scenarios.

- **Intermediate Complexity:** Another test involved a 98-digit number $N=2^{322.6}$, where DYNEX's capability was again prominently displayed, solving the problem in 1.23 seconds. Conversely, the classical solver required 29 minutes to achieve a solution, further emphasizing the DYNEX platform's superior time efficiency across varying levels of problem complexity.

These comparative results (Figures 1-3) provide a clear visualization of the time efficiency advantage that the DYNEX Neuromorphic Network holds over traditional solvers. It is this capability to dramatically reduce computational time, especially for large-scale and complex problems, that positions the DYNEX platform as a transformative force in the field of computational mathematicsand cryptography.

8. Conclusion

This document presented a comprehensive overview of the transition from HUBO to QUBO models, with a special emphasis on prime factorization problems and their computation on the DYNEX Neuromorphic Network. Our findings demonstrate that neuromorphic computing, particularly the DYNEX platform, offers significant advantages in terms of computational speed and efficiency compared to classical prime factor solvers [7]. The ability of the DYNEX network to solve a 201-digit prime factorization problem in mere seconds is a testament to its potential to revolutionize the field of computational mathematics and cryptography.

The advent of quantum computing poses both challenges and opportunities. While it threatens the security of current cryptographic algorithms, it also spurs the development of quantum-resistant algorithms [1]. The DYNEX Neuromorphic Network is poised to play a pivotal role in this new era, providing a platform that can evolve with the computational demands of quantum algorithms and offer robust security solutions.

As we look to the future, the continued advancement of DYNEX Neuromorphic computing technologies is expected to accelerate. The DYNEX Neuromorphic Network, with its innovative approach and scalability, is well-positioned to lead the charge in developing quantum-resistant cryptographic algorithms. This promises not only to safeguard digital security in the age of quantum computing but alsoto unlock new potential in various domains that rely on complex computation.

Further research and development will focus on optimizing the neuromorphic computing models, enhancing the speed and accuracy of computations, and exploring new applications beyond prime factorization. As the landscape of computing evolves, the DYNEX Neuromorphic Network will continue toadapt and advance, ensuring that it remains at the forefront of this technological revolution.

References

1. Jun K, Lee H. HUBO and QUBO models for prime factorization. Sci Rep. 2023;13:10080.

2. Rieffel EG, Polak WH. Quantum Computing: A Gentle Introduction. MIT Press, Massachusetts, USA. 2014.

3. Lu Y, Sigov A, Ratkin L, et al. Quantum computing and industrial information integration: a review. J Ind Inf Integration. 2023;35:100511.

4. Zhu Y, Zhu Y, Mao H, et al. Recent advances in emerging neuromorphic computing and perception devices. J Phys D: Appl Phys. 2021;55:053002.

5. https://docs.ocean.dwavesys.com/en/stable/

6. Wang B, Hu F, Yao H, et al. Prime factorization algorithm based on parameter optimization of Ising model. Sci Rep. 2020;10:7106.

7. https://docs.dynexcoin.org/

8. Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM. 1978;21:120-6.

9. Shor PW. Algorithms for quantum computation: discrete logarithms and factoring. Proceedings 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA. 1994.

10. Khatarkar S, Kamble R. A survey and performance analysis of various RSA based encryption techniques. Int J Comput Appl. 2015;114:30-3.

11. Lindsay JR. Surviving the quantum cryptocalypse. Strategic Studies Quarterly. 2020;14:49-73.

12. Zeng GQ, Lu KD, Dai YX, et al. Binary-coded extremal optimization for the design of PID controllers. Neurocomputing. 2014;138:180-8.

13. Bennell C, Emeno K, Snook B, et al. The precision, accuracy and efficiency of geographic profiling predictions: a simple heuristic versus mathematical algorithms. Crime Mapping: A Journal of Research and Practice. 2009;1:65-84.

14. https://github.com/dynexcoin