# Polynomial-time Solver of Tridiagonal QUBO, QUDO and Tensor QUDO problems with Tensor Networks

Alejandro Mata Ali*
*i3B Ibermatica, Quantum Development Department, Paseo Mikeletegi 5, 20009 Donostia, Spain*

Iñigo Perez Delgado† and Marina Ristol Roura‡
*i3B Ibermatica, Parque Tecnológico de Bizkaia, Ibaizabal Bidea, Edif. 501-A, 48160 Derio, Spain*

Aitor Moreno Fdez. de Leceta§
*i3B Ibermatica, Unidad de Inteligencia Artificial,*
*Avenida de los Huetos, Edificio Azucarera, 01010 Vitoria, Spain*
(Dated: July 15, 2025)

We present a quantum-inspired tensor network algorithm for solving tridiagonal Quadratic Unconstrained Binary Optimization (QUBO) problems and quadratic unconstrained discrete optimization (QUDO) problems. We also solve the more general Tensor quadratic unconstrained discrete optimization (T-QUDO) problems with one-neighbor interactions in a lineal chain. This method provides an exact and explicit equation for these problems. Our algorithms are based on the simulation of a state that undergoes imaginary time evolution and a Half partial trace. In addition, we address the degenerate case and evaluate the polynomial complexity of the algorithm, also providing a parallelized version. We implemented and tested them with other well-known classical algorithms and observed an improvement in the quality of the results. The performance of the proposed algorithms is compared with the Google OR-TOOLS and dimod solvers, improving their results.

Keywords: Tensor networks, Quantum-inspired, Combinatorial optimization, Quantum computing

## I. INTRODUCTION

Quadratic Unconstrained Binary Optimization (QUBO) [1] problems are a type of combinatorial optimization problem that lies at the intersection of quantum computing and optimization theory. These problems are characterized by their ability to represent a wide variety of complex challenges and their use in industrial and applied fields such as logistics [2, 3], engineering [4], physics [1], biology [5], and economics [6]. In a QUBO problem, one seeks to find an assignment of binary values (0 or 1) to a set of decision variables such that the quadratic function of these variables is minimized. This quadratic function, also known as a cost or energy function, can represent constraints and objectives of a specific problem. Quadratic Unconstrained Discrete Optimization (QUDO) problems are a generalization of the QUBO problems, allowing the variables to take a larger number of integer values and can be translated to QUBO problems with a logarithmic amount of binary QUBO variables for each discrete QUDO variable. In order to represent more complex problems, other formalisms have been developed. The most well-known generalization is the Higher-Order Unconstrained Binary Optimization (HOBO) problem, which has a cost function that involves products of more than two binary variables. Another novel approach is the Tensor Quadratic Unconstrained Discrete Optimization (T-QUDO) problem [7], which uses tensor positions instead of products to represent more naturally certain types of problem. Both problems have a degree of complexity too high to be efficiently solved classically [8], except in certain cases [9, 10]. As a result, they are often tackled using approximate or heuristic methods, such as genetic algorithms [11], linear programming [12] or digital annealing [13].

QUBO problems are particularly interesting and relevant in the era of quantum computing because of their ability to take advantage of the properties of quantum computing. This is due to the equivalence between the QUBO problems and the Ising model, allowing algorithms such as the Quantum Approximate Optimization Algorithm (QAOA)[14] to solve them. It has been applied in digital quantum computing [15], quantum annealing [16], and hybrid algorithms [2, 3]. QUDO problems are also compatible with quantum computing [17], since it is possible to make use of qudits or groups of qubits that simulate to be one qudit, or transform the QUDO into a QUBO problem.

However, due to the current state of quantum hardware, noisy and with small capacity [18], and its availability, the field of quantum-inspired technologies has gained importance. These classical technologies involve taking advantage of certain quantum properties to accelerate calculations. One of the most important of these is tensor networks [19, 20], which use algebraic mathematics of quantum systems to simulate them classically and extract certain properties of the simulated systems. They

* alejandro.mata.ali@gmail.com
† iperezde@ayesa.com
‡ mristol@ayesa.com
§ aitormoreno@lksnext.com

can implement operations that would not be possible in quantum systems, such as forced post-selection or the application of non-unitary operators. This technology makes possible the compression of information, both for classical and quantum systems, and has been applied to machine learning [21], large language model [22] compression, and quantum algorithm simulation [23].

For solving 1D QUBO problems, there are algorithms in tensor networks, such as [12]. There are also other techniques for more general combinatorial optimization problems [24, 25]. However, these have a high computational complexity that may make them not optimal for large instances.

In this paper, we will explore an efficient tensor network algorithm to solve general QUBO, QUDO, and Tensor QUDO problems with one-neighbor interactions in a lineal chain. We call it the *Lineal Chain Quadratic Problem Tensor Network Solver* (LCQPTNS). The connection of these problems with Ising models could lead to applications in quantum ground-state simulation with the same formalism. It will consist of a sequential tensor network contraction algorithm that simulates an imaginary time evolution and a partial trace. We will also analyze its applicability to degenerate cases and see its computational complexity. Our main contributions are

- Provide a novel methodology for solving 1D combinatorial problems.
- Provide a novel algorithm for solving general QUBO, QUDO and Tensor QUDO problems with one-neighbor interactions in a lineal chain. To the best of our knowledge, this is the first algorithm to address this exact problem directly.
- Provide the first parallelized algorithm for MeLoCoToN problem resolution applications.
- Provide an explicit and exact equation that returns the solution for the three problems.
- Analyze the performance of the algorithm, theoretically and empirically, and compare it with other algorithms.
- Provide a Python implementation of the algorithms.

This work is structured as follows. First, we describe the problems we want to solve in Sec. II and we present a brief background on the state-of-the-art in solving them. Then, we introduce the three novel algorithms in Sec. III. Finally, we performed several experiments in Sec. IV to analyze its performance.

All code is publicly available on the GitHub repository https://github.com/DOKOSTAYOS/Lineal_chain_QUBO_QUDO_TensorQUDO

## II. DESCRIPTION OF THE PROBLEM

A general QUBO problem can be expressed by a quadratic cost function to minimize using a vector $\vec{x}$ of $N$ binary components. That is, we look for an optimal $\vec{x}_{\text{opt}}$ such that

$$\begin{aligned} \vec{x}_{\text{opt}} &= \arg\min_{\vec{x}} C(\vec{x}), \\ x_i &\in \{0,1\}, \quad i \in [0, N-1] \\ C(\vec{x}) &= \sum_{\substack{i,j=0 \\ i\leq j}}^{N-1} w_{ij} x_i x_j, \end{aligned} \tag{1}$$

where $w_{ij}$ are the elements of the weight matrix $w$ of the problem and $C(\cdot)$ is the cost function of the problem. The diagonal elements $w_{ii}$ are the local terms, and the non-diagonal elements $w_{ij}$ are the interaction terms.

In a QUDO case, the problem is analogous, changing that the components of $\vec{x}$ will be integers in a certain range, not only 0 or 1. This is

$$\begin{aligned} \vec{x}_{\text{opt}} &= \arg\min_{\vec{x}} C(\vec{x}) \\ x_i &\in \{0,1,\dots,D_i-1\}, \quad i \in [0, N-1] \\ C(\vec{x}) &= \sum_{\substack{i,j=0 \\ i\leq j}}^{N-1} w_{i,j} x_i x_j + \sum_{i=0}^{N-1} d_i x_i, \end{aligned} \tag{2}$$

being $d_i$ the elements of the cost vector $\vec{d}$ of the problem and $D_i$ the number of possible values of the $i$-th variable. QUDO problems can be transformed into QUBO problems transforming $x_i$ variables into a set of $s_{i,k}$ variables

$$x_i = \sum_{k=0}^{\log_2 D_i - 1} 2^k s_{i,k}, \tag{3}$$

if $\log_2 D_i$ is an integer.

In a Tensor QUDO case, the cost function of the problem now is formulated as the sum of the elements of a tensor, selected by the solution vector. This is

$$\begin{aligned} \vec{x}_{\text{opt}} &= \arg\min_{\vec{x}} C(\vec{x}), \\ x_i &\in \{0,1,\dots,D_i-1\}, \quad i \in [0, N-1] \\ C(\vec{x}) &= \sum_{\substack{i,j=0 \\ i\leq j}}^{N-1} w_{i,j,x_i,x_j} \end{aligned} \tag{4}$$

where $w_{i,j,x_i,x_j}$ are the elements of the weight tensor $w$ of the problem, which depends on the value of the variables and their positions in the solution. Both QUBO and QUDO problems are particular cases of Tensor QUDO problems.

A special case of interest is the case of nearest-neighbor interaction in a linear chain, which can be understood as the Ising model in one dimension. In this problem, each

variable interacts only with the one it has before and with the one it has just after. Therefore, our QUBO and QUDO problems cost function simplifies to

$$C(\vec{x}) = \sum_{i=0}^{N-1} (w_{i,i}x_i^2 + d_i x_i) + \sum_{i=0}^{N-2} w_{i,i+1}x_i x_{i+1}, \quad (5)$$

which implies that $w$ is a tridiagonal matrix

$$w = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 & 0 \\ w_{21} & w_{22} & w_{23} & 0 & 0 \\ 0 & w_{32} & w_{33} & w_{34} & 0 \\ 0 & 0 & w_{43} & w_{44} & w_{45} \\ 0 & 0 & 0 & w_{54} & w_{55} \end{bmatrix}. \quad (6)$$

In the Tensor QUDO case, the analogous problem would be

$$C(\vec{x}) = \sum_{i=0}^{N-1} w_{i,i,x_i,x_i} + \sum_{i=0}^{N-2} w_{i,i+1,x_i,x_{i+1}} \quad (7)$$

which could be considered a kind of shortest path problem without restrictions, where the distances change with every time step.

There exist several algorithms to solve QUBO problems, from classical ones, such as Spiking Neural Networks [10–12, 26], to quantum computing ones, such as QAOA [14]. However, in the state-of-the-art, there are no prior algorithms that address the lineal chain problem exactly and efficiently. The most efficient algorithm for this 1D problem is presented in [12], but even in the most simple case, it runs as $O(n^{16})$. And this algorithm also scales exponentially with $D_i$ for the QUDO and Tensor QUDO problems, so it is not efficient in these cases. Taking into account that the general QUBO problem is an NP-Hard problem, all known exact algorithms to solve it have exponential complexity.

In heuristic algorithms to approach this type of 1D problem, the Density Matrix Renormalization Group (DMRG) [27] is the most well-known algorithm. However, this method is only effective if the bond dimension does not scale exponentially. Its usual complexity in QUDO cases would be $O(ND^3\chi^3)$ for each sweep, being $D$ the typical dimension of the variables and $\chi$ the required bond dimension, and could be extremely low for large systems. Additionally, this algorithm is not designed for Tensor QUDO problems.

## III. TENSOR NETWORK ALGORITHMS

In this section we introduce the new tensor network algorithms for solving QUBO, QUDO and Tensor QUDO problems in a lineal chain with one-neighbor interaction efficiently. First, in Ssec. III A we introduce the QUBO algorithm, as a first and simpler version. Secondly, in Ssec. III B we introduce how to generalize it for QUDO problems. Third, in Ssec. III C we generalize it for Tensor QUDO problems. Then, we discuss how to optimize the tracing method in Ssec. III D and to address the degenerate case in Ssec. III E. Finally, in Ssec. III F we analyze the computational complexity of the algorithms execution.

### A. Tridiagonal QUBO tensor network solver

First we will create a solver for the tridiagonal QUBO problem and then we will see how to generalize it to the QUDO problem. We use a modified version of the method [25], and it is the first algorithm to use the MeLoCoToN formalism [28]. Our algorithm can be summarized in the following theorem.

**Theorem 1** *Given a QUBO problem described by a tridiagonal $N \times N$ weight matrix $w$ and $N$ indeterminate binary variables, we can determine an approximate optimal solution in $O(N)$ time, which is the optimal in the limit as $\tau \to \infty$.*

For this, we will explain the basics of the algorithm with the tensor network of Fig. 1 a, which mimics the shape of a quantum circuit, with each horizontal line being the timeline of a qubit and the vertical lines controlling operations between them. The '+' nodes represent a qubit in uniform superposition $(1, 1)$ and the $T$ nodes represent the imaginary time evolution that depends on the state of the two neighboring qubits.

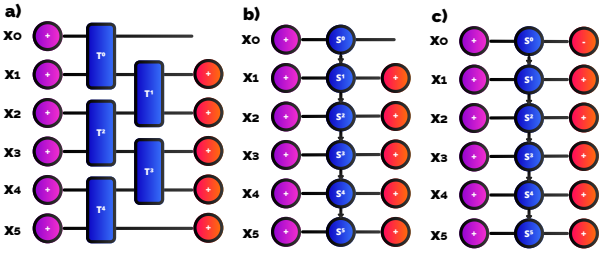


FIG. 1. Tensor networks solving the tridiagonal QUBO problem and the one-neighbor QUDO problem. a) Bilocal version, b) Matrix Product Operator (MPO) version, c) Matrix Product Operator (MPO) version for explicit equation.

The '+' nodes represent the state of each variable $x_i$. It can be visualized as the initialization of a quantum circuit. By performing the tensor product with all these tensors, each in uniform superposition, we will have uniform superposition of all $2^N$ combinations. That is, a vector of $2^N$ ones.

The goal is for our tensor layer $T$ to encode the state in our tensor network as

$$|\psi\rangle = \sum_{\vec{x}} e^{-\tau C(\vec{x})} |\vec{x}\rangle, \quad (8)$$

so that the combination $\vec{x}$ with the lowest cost has an exponentially larger amplitude than the other combinations. In tensor terms, the tensor represented by this

tensor network is

$$\mathcal{T}_{\vec{x}} = e^{-\tau C(\vec{x})}. \tag{9}$$

The $T$ tensors will be tensors with four 2-dimensional indices $i, j, \mu, \nu$ (Fig. 2) whose non-zero elements correspond to cases where $\mu = i$ and $\nu = j$. This means that the state of the first variable enters at index $i$ and exits at index $\mu$ and the state of the second variable enters at index $j$ and exits at index $\nu$. Using the sparse logical notation introduced in [28], the non-zero elements are

$$\begin{gathered} T^n_{2\times 2\times 2\times 2}, \\ \mu = i, \nu = j, \\ T^n_{ij\mu\nu} = e^{-\tau(w_{n,n+1}ij + w_{n,n}i)}, \end{gathered} \tag{10}$$

$$\begin{gathered} T^{N-2}_{2\times 2\times 2\times 2}, \\ \mu = i, \nu = j, \\ T^{N-2}_{ij\mu\nu} = e^{-\tau(w_{N-2,N-1}ij + w_{N-2,N-2}i + w_{N-1,N-1}j)}, \end{gathered} \tag{11}$$

where $\tau$ is a decay hyperparameter, related to the evolution in imaginary time, and $T^n$ is the $T$-tensor which connects the $n$-th and $n+1$-th variables.

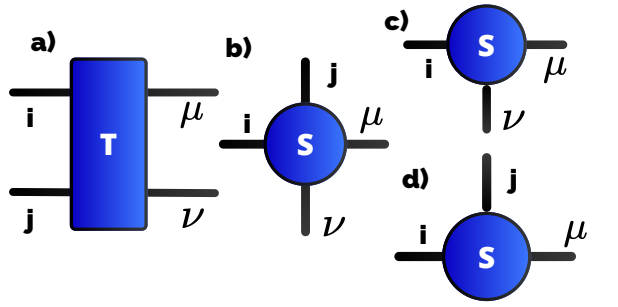


FIG. 2. Tensor indices. a) $T$ tensor, b) First $S$ tensor, c) Intermediate $S$ tensor, d) Final $S$ tensor.

Since the resulting tensor has $2^N$ components, we cannot simply look at which component has the largest amplitude. We will extract its information in a more efficient way. We assume that the amplitude of the lowest-cost combination is sufficiently larger than the amplitudes of the other combinations. This is a reasonable assumption because if we increase $\tau$, the combinations will change their amplitudes exponentially differently. In the infinite limit, only the relative amplitude of the optimal solution will remain. From this tensor we will obtain the value of $x_0$ associated with the optimal combination. If there is a combination with a sufficiently higher amplitude, adding the amplitudes of all combinations with $x_0 = 0$ and with $x_0 = 1$ separately, the main contribution will be the one with the highest amplitude. We will call this operation *Half Partial Trace* with $x_0$ free, which returns a vector $P^{x_0}$ with components.

$$P_i^{x_0} = \sum_{\substack{\vec{x} \\ x_0 = i}} e^{-\tau C(\vec{x})}. \tag{12}$$

This partial trace is made by connecting a '+' tensor to each variable at the output of the $T$ tensor layer, except for the $x_0$ variable. Taking the limit of large $\tau$, assuming no degeneracy,

$$\lim_{\tau\to\infty} \frac{P_i^{x_0}}{\sum_i P_i^{x_0}} = \lim_{\tau\to\infty} \frac{\sum_{\substack{\vec{x} \\ x_0 = i}} e^{-\tau C(\vec{x})}}{\sum_i \sum_{\substack{\vec{x} \\ x_0 = i}} e^{-\tau C(\vec{x})}}. = \delta_{i,(x_{opt,0})} \tag{13}$$

Out of the limit, if the combination with the highest amplitude has $x_0 = 0$, then $P_0^{x_0} > P_1^{x_0}$, and in the opposite case $P_0^{x_0} < P_1^{x_0}$. This also means that, if we multiply the resulting vector by a *minus vector* $(-1, 1)$, if the correct value is $x_0 = 0$, so $P^{x_0} = (1, 0)$, then the contraction of the tensor network is $-1$, and if it is $x_0 = 1$, then it results in $+1$. So, the correct value can be expressed as

$$x_0 = H(\Omega_0(\tau)) \tag{14}$$

namely $H(\cdot)$ the Heaviside step function and $\Omega_0(\tau)$ the tensor network described with the minus vector in the 0-th position for a $\tau$ value. This is represented in Fig. 1 c for six variables.

We can optimize the contraction of the tensor network by defining it as shown in Fig. 1 b, where we replace the $T$-tensors by a Matrix Product Operator (MPO) layer of $S$-tensors. These tensors perform exactly the same function, sending signals up and down through their vertical bond indices. All of their indices are of dimension 2. In the figure, the arrows indicate the flow of information through the nodes of the layer. These will tell adjacent $S$ tensors about their associated variable state and, depending on the signal they receive from the previous $S$ tensor and their own variable, apply a certain imaginary time evolution. This tensor network is much easier and more straightforward to contract. We call $S^n$ the $S$-tensor connected with the variable $x^n$.

Therefore, the non-zero elements of the $S$-tensors will be those where $\mu = i$ for $S^{N-1}$ and $\mu = \nu = i$ for the others. Their values will be

$$\begin{gathered} S^0_{2\times 2\times 2}, \\ \mu = \nu = i, \\ S^0_{i\mu\nu} = e^{-\tau(w_{0,0}i)}, \end{gathered} \tag{15}$$

$$\begin{gathered} S^n_{2\times 2\times 2\times 2}, \\ \mu = \nu = i, \\ S^n_{ij\mu\nu} = e^{-\tau(w_{n-1,n}ij + w_{n,n}i)}, \end{gathered} \tag{16}$$

$$\begin{gathered} S^{N-1}_{2\times 2\times 2}, \\ \mu = i, \\ S^{N-1}_{ij\mu} = e^{-\tau(w_{N-2,N-1}ij + w_{N-1,N-1}i)}. \end{gathered} \tag{17}$$

With the corresponding tensor network, we first determine the variable $x_0$, then follow these steps to get the other components:

1. Perform the algorithm for the $x_0$ component.
2. For $n \in [1, N-2]$:
   - Carry out the same algorithm, but eliminating the previous $n-1$ variables. Change the new tensor $S^0$ so that its components match those of the tensor $S^1$ from the previous step, with its index $j$ set to $x_{n-1}$. This means
   $$S^0_{i\mu\nu} = S^{1,\text{previous}}_{i,x_{n-1},\mu,\nu}. \tag{18}$$
   The result of the algorithm will be $x_n$.
3. For $x_{N-1}$, we use the classical comparison:
$$x_{N-1} = H(-(w_{N-1,N-1} + w_{N-2,N-1}x_{N-2})). \tag{19}$$

The progressive reduction of the represented variables is due to the fact that if we already know the solution, we can consider the rest of the unknown system and introduce the known information into the system. The redefinition of the tensor $S^0$ in each step allows us to consider the result of the previous step in obtaining the costs of the variable pair to be determined and the previous one.

To obtain the exact explicit equation, we will omit the variables reduction to simplify the explanation. If we have a tensor network of layers '+' and $S$, we can extract the $n$-th variable connecting the '+' nodes for all the variables, but the $n$-th one, which will be connected to a minus vector. We call the resulting tensor network $\Omega_n(\tau)$. We know that in the limit this algorithm is exact if we have no degeneracy, so we can express

$$x_n = \lim_{\tau\to\infty} H(\Omega_n(\tau)). \tag{20}$$

This equation is exact because all the arguments previously shown and explicit because the solution does not depend on the own solution we want. So, the Heavyside step function of this tensor network is the equation for the solution of the problem. Then, this is an exact and explicit equation to solve the tridiagonal QUBO problem.

### B. Tridiagonal QUDO problem

In the QUDO problem, we can use the same tensor network structure. We will only have to change our binary variable formalism to a discrete variable formalism and allow the indices of the tensor network in Fig. 1 to have the dimension required for each variable. From now on, the variable $x_i$ will have $D_i$ possible values. Our algorithm can be summarized in the following theorem.

**Theorem 2** *Given a QUDO problem described by a tridiagonal $N \times N$ weight matrix $w$ and $N$ indeterminate binary variables of $D$ possible values, we can determine an approximate optimal solution in $O(ND^2)$ time, which is optimal in the limit as $\tau \to \infty$.*

Each '+' tensor will now have $D_i$ components, depending on the variable they represent, all only with ones. The same is true for those we use to make the partial trace. The construction is the same, but the new $S$-tensor definitions are

$$\begin{gathered} S^0_{D_0\times D_0\times D_0}, \\ \mu = \nu = i, \\ S^0_{i\mu\nu} = e^{-\tau(w_{0,0}i^2 + d_0 i)}, \end{gathered} \tag{21}$$

$$\begin{gathered} S^n_{D_n\times D_{n-1}\times D_n\times D_n}, \\ \mu = \nu = i, \\ S^n_{ij\mu\nu} = e^{-\tau(w_{n-1,n}ij + w_{n,n}i^2 + d_n i)}, \end{gathered} \tag{22}$$

$$\begin{gathered} S^{N-1}_{D_{N-1}\times D_{N-2}\times D_{N-1}}, \\ \mu = i, \\ S^{N-1}_{ij\mu} = e^{-\tau(w_{N-2,N-1}ij + w_{N-1,N-1}i^2 + d_{N-1}i)}. \end{gathered} \tag{23}$$

As in the QUBO case, the $S$-tensors receive the state of the adjacent variable through the index $j$ and send theirs through the index $\nu$.

The rest of the algorithm is the same until the value extraction. If we contract this tensor network analogously to the QUBO case, we obtain a vector $P^{x_0}$ of dimension $D_0$. From this, we can extract the optimal value of $x_0$ by looking for the largest component. That is, if the vector obtained by the tensor network were $(2, 4, 27, 2, 0, 1)$, the correct value for $x_0$ would be 2. To obtain the other variables, we will perform exactly the same process that we have explained for the QUBO case. We will change the last step to a comparison that gives us the value of $x_{N-1}$, which will result in a lower cost based on the value of $x_{N-2}$ we already have.

To obtain the explicit equation, in the index of the variable we want to determine, we cannot simply connect a minus vector because the dimensions do not fit. However, we can binarize the value of $x_n$ into a set of binary variables $x_{n,m}$. This means that instead of connecting a minus vector, we connect a *bit-selector vector* $B$. To exemplify this vector, for a dimension four case, it is $(-1, 1, -1, 1)$ for the first bit determination and $(-1, -1, 1, 1)$ for the second one. In general, to determine the $m$ bit having a dimension $D$, the $B^{D,m}$ vector $i$-th component is defined as

$$B^{D,m}_i = (-1)^{b(i)_m+1}, \tag{24}$$

being $b(i)$ the binary vector of the binarization of $i$.

This means that to determine the $m$-th bit of the $x_n$ variable, the equation is the following.

$$x_{n,m} = \lim_{\tau\to\infty} H(\Omega_{n,m}(\tau)), \tag{25}$$

being $\Omega_{n,m}(\tau)$ the tensor network defined for the problem, Half Partial traced for all the variables, but $n$-th one, which is connected with $B^{D_n,m}$. This is an exact and explicit equation to solve the tridiagonal QUDO problem.

### C. One-neighbor Tensor QUDO problem

In this case, we are going to solve the lineal chain one-neighbor Tensor QUDO problem using an extended version of the tridiagonal QUDO algorithm. Here, we only need to change the value of the elements of the $S$-tensors.

Our algorithm can be summarized in the following theorem.

**Theorem 3** *Given a Tensor QUDO problem described by a lineal chain one-neighbor $N \times N \times D \times D$ weight tensor $w$ and $N$ indeterminate variables with $D$ possible values, we can determine an approximate optimal solution in $O(ND^2)$ time, which is optimal in the limit as $\tau \to \infty$.*

Once again, the elements of the $S$-tensors are non-zero when $\mu = i$ for $S^{N-1}$, and $\mu = \nu = i$ for the others. They are

$$\begin{gathered} S^0_{D_0 \times D_0 \times D_0}, \\ \mu = \nu = i, \\ S^0_{i\mu\nu} = e^{-\tau w_{0,0,i,i}}, \end{gathered} \tag{26}$$

$$\begin{gathered} S^n_{D_n \times D_{n-1} \times D_n \times D_n}, \\ \mu = \nu = i, \\ S^n_{ij\mu\nu} = e^{-\tau (w_{n-1,n,i,j} + w_{n,n,i,i})}, \end{gathered} \tag{27}$$

$$\begin{gathered} S^{N-1}_{D_{N-1} \times D_{N-2} \times D_{N-1}}, \\ \mu = i, \\ S^{N-1}_{ij\mu} = e^{-\tau (w_{N-2,N-1,i,j} + w_{N-1,N-1,i,i})}. \end{gathered} \tag{28}$$

The rest of the algorithm is the same as in the QUDO problem, and the exact and explicit equation is constructed in the same way.

### D. Tracing optimization

One of the biggest problems we may encounter is choosing a value of $\tau$ large enough to distinguish the optimal combination but not so large that the amplitudes go to zero. For this reason, in practice, it is advisable to rescale the $w$ matrix before starting so that the minimum cost scale does not vary too much from problem to problem. In this way, we can leave $\tau$ constant. A good practice to ensure this may be that the maximum cost we can obtain is on the order of 1 and the minimum on the order of -1 or 0, depending on the characteristics of the problem.

In addition, an effective way to modify the general algorithm is to initialize in a superposition state with complex phases between the base combinations instead of initializing with a superposition with the same phase. This allows us that, when we perform the Half Partial Trace, instead of summing all amplitudes in the same direction, they will be summed as 2-dimensional vectors. This facilitates the suboptimal states to be damped against each other, allowing the maximum to be seen better.

We add these phases by initializing the '+' tensors in $(e^{2\pi i \cdot 0 \frac{1}{D_n}}, e^{2\pi i \cdot 1 \frac{1}{D_n}}, e^{2\pi i \cdot 2 \frac{1}{D_n}}, \ldots, e^{2\pi i \cdot (D_n - 1) \frac{1}{D_n}})$ instead of in $(1, 1, \ldots, 1)$. Thus, each possible combination has its own associated phase. We call it the *Humbucker* method, inspired in the noise cancellation for guitar coils. To improve performance, we can add a small random factor to each combination phase.

In this case, we only need to change the definition of $P^{x_0}$ to

$$P_i^{x_0} = \left|\left| \sum_{\substack{\vec{x} \\ x_0 = i}} e^{i\gamma_{\vec{x}}} e^{-\tau C(\vec{x})} \right|\right|, \tag{29}$$

being $\gamma_{\vec{x}}$ the phase of the $\vec{x}$ state.

Another problem that can arise is to have an $N$ such that when adding the $D^N$ damped states, we exceed the precision of the numbers we are using, obtaining divergences in the trace vector. Related to this problem, we have the possibility that by multiplying so many nodes, we obtain all 0 values due to lack of precision in our numbers. To deal with this, we can choose to divide the components of the $n$ initialization '+' nodes of step $n$ so that we try to make the sum of the components of the final trace vector near $D_n$. That is, we will try to have a 1-norm

$$\sigma_n = \sum_i P_i^{x_0} = D_n. \tag{30}$$

We choose $D_n$ as the target value to account for the possibility of having large differences in $D$ between the problems and the variables. To do so, we will make each initialization node '+' at step $n$ multiply its components by

$$\left(\frac{1}{D_n}\right)^{\frac{1}{N-n}} \xi_n = \left(\frac{1}{D_n}\right)^{\frac{1}{N-n}} \xi_{n-1}^{\frac{N-n+1}{N-n}} \left(\frac{1}{\sigma_{n-1}}\right)^{\frac{1}{N-n}} \tag{31}$$

where $\xi_n$ is the factor that accounts for how much $\left(\frac{1}{\sigma_{n-1}}\right)^{\frac{1}{N-n}}$ would be if we did not apply this normalization, with $\xi_0 = 1$.

By doing this, we will be normalizing the states so that the state after the time evolution will have a 1-norm closer to a controlled value, avoiding divergences. Another possibility is to implement the normalization algorithm presented in [29].

### E. Degenerate case

Until now, we have considered a non-degenerate case. However, in the degenerate case, where we have more than one optimal combination, we have more peaks of similar amplitude. They are not exactly equal, because of the contribution of residual states. In the phased method, we can have the situation where the two peaks have opposite phases, so that they cancel out and we cannot see the optimum. For this reason, we recommend that the phased version should not be used in case of a possible degeneration of the problem.

Our method without phases allows us to avoid the degeneracy problem, because if at any step of the process we have two peaks with two different values of $x_n$, we will choose one of them and the rest of the combination we obtain will be the one associated exactly with the $x_n$ we have chosen, avoiding the degeneracy problem. In addition, if we want the other states of the degeneracy, we can do the same process again, but choosing the other high-amplitude component instead.

### F. Complexity analysis

Before analyzing its computational complexity, we have to emphasize that the tensor network can be further optimized. We take into account the fact that the contraction of the '+' tensors with the $S$ tensors only implies that the resulting tensors will be the same as the $S$ tensors that created them, but by simply eliminating the $i$ index. For the traced variables, it also means removing the $\mu$ index, but in the last one, which implies the sum of the values for all the $i$ values. So, the optimal tensor network would be exactly the same as in Fig. 1 b, but eliminating the first and last layers of the '+' tensors and their associated indices in the $S$ tensors. This tensor network is the one in Fig. 3 a.

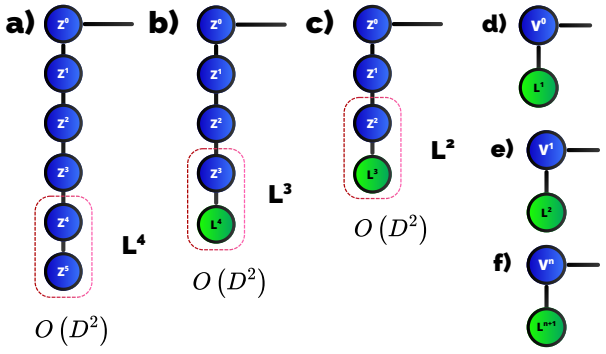


FIG. 3. Contraction scheme for the efficient tensor network. We have to repeat this process $N-1$ times to obtain a component. a) Partial trace of the last variable. b) Partial trace of the penultimate variable. c) Contraction of the last variable with the penultimate one. d)

The definition of $Z$-tensors in the Tensor QUDO case is

$$Z^0_{D_0\times D_0},\quad \nu = i,\quad Z^0_{i\nu} = e^{-\tau w_{0,0,i,i}}, \tag{32}$$

$$Z^n_{D_{n-1}\times D_n},\quad Z^n_{ij} = e^{-\tau(w_{n-1,n,i,j}+w_{n,n,i,i})}, \tag{33}$$

$$Z^{N-1}_{D_{N-1}},\quad Z^{N-1}_j = \sum_{i=0}^{D_{N-1}} e^{-\tau(w_{N-2,N-1,i,j}+w_{N-1,N-1,i,i})}, \tag{34}$$

and analogous for the QUBO and QUDO problems. The complexity of generating these tensors is $O(D^2)$ for each one, the same for their space complexity. Taking into account that they are $N$ different, we have a computational and space complexity of $O(ND^2)$ for the creation and storage of tensors. However, if we create them just before using them, we can store only one matrix at each step, so the complexity would be $O(D^2)$.

The computational complexity of contracting each of the tensor networks for a QUDO problem with $N$ variables that can take $D$ values is $O(ND^2)$, because it multiplies $N$ matrices $D\times D$ by a vector. This is due to having to apply the contraction scheme in Fig. 3. We have to repeat it $N-1$ times to determine the $N$ variables, so the total computational complexity of the algorithm is $O(N^2D^2)$, the space complexity remains in $O(D^2)$, and with solution storage it is $O(N+D^2)$. However, storage of the $w$ tensor of Tensor QUDO requires $O(ND^2)$ space complexity, so that is the space complexity in this case. In the QUBO case, the computational complexity is $O(N^2)$.

If we have $M$ degenerate solutions, to obtain all of them, we will only have to apply the process $M$ times, so that we have a complexity of $O(MN^2D^2)$.

However, we can improve execution time by reusing intermediate computations. To do so, when calculating the result of the first variable, we store the intermediate tensors $L^n$ that we see in Fig. 3 b and c. In this way, since we only need to change the $n$-th node for a new $V^n$ tensor, as we explained for the iterative method, the rest of the tensor network, stored in the tensor $L^{n+1}$ will be the same. Therefore, we will only need to multiply the new $V^n$ tensor by the tensor $L^{n+1}$ that we had stored from the first iteration.

Since each of these multiplications has a cost $O(D)$, because the matrix is diagonal, and we have to do $O(N)$, the total cost of determining the variables from 1 to $N-2$ will be $O(ND)$. Since the cost to determine the first is $O(ND^2)$, we will have a total computational complexity of $O(ND^2)$. The storage of the $L$-tensors in the first contraction requires space complexity $O(ND)$, so the total space complexity is $O(ND+D^2)$. Again, for Tensor

QUDO the space complexity is $O(ND^2)$. In the QUBO case, the computational and space complexity is $O(N)$. If we have $M$ degenerate solutions, to obtain all of them, we will only have to apply the process $M$ times, so that we have a complexity of $O(MND^2)$. Thus, it is determined that the algorithm has a linear cost in the number of variables and a quadratic cost in the variable dimensionality.

The algorithm can be parallelized, even without an improvement in computational complexity. For simplicity, we assume that we have enough units for computation that can compute in parallel, so we will analyze the parallel runtime with infinite processors. From this point on, we call computational complexity the main component of this parallel runtime. Our intention is to determine the vector after the Half Partial Trace without knowing the previous variables results. That is, we will compute these vectors in the first iteration of the tensor network contraction for every possible value of the previous variable and, after determining it, choose the corresponding vector. We will understand it better with the explicit algorithm. We define *BigUnit* as a set of units that can compute in parallel the vector-matrix products, and *SmallUnit* as the set of units that compose a BigUnit that computes in parallel each element of the result of the vector matrix product. The algorithm is as follows:

1. We create each $Z$ tensor. Every tensor is computed in a different BigUnit, allowing us to compute them in parallel, and each element is computed in a different SmallUnit in its corresponding BigUnit, allowing their parallel computation. These are all the tensors required in the definition, and can be computed in a $O(1)$ time, but the last one that requires the summation of $D$ elements, taking $O(\log_2(D))$ time making pair sums. The computational complexity is $O(\log_2(D))$ and the space complexity is $O(ND^2)$. However, space complexity can be reduced to $O(N + D^2)$ if, instead of generating all $Z$-tensors in the first step, we only generate them just before contracting them.

2. At the same time, in $D + 1$ different BigUnits, we compute:

   - We contract $Z^{N-1}$ with $Z^{N-2}$ to get $L^{N-2}$ in a BigUnit, computing each of its elements with a SmallUnit. Each element is computed in parallel and requires sum $D$ elements, which means it has computational complexity $O(\log_2(D))$ with pairs summation and space complexity $O(D^2)$.
   - The contraction of $Z^{N-1}$ with $V^{N-2}$ for every possible value of $x_{N-3}$ to obtain all the vectors $P^{x_{N-2},x_{N-3}}$. Every contraction has computational complexity $O(1)$, because $V^{N-2}$ is a diagonal matrix, and they are performed in parallel in different BigUnits. Its space complexity is $O(D^2)$.
   - Now, for each value of $x_{N-3}$, we save the position of the largest component, obtaining the vector $X_{N-2}$. This vector component $X_{N-2,i}$ has the correct values for $x_{N-2}$ if the correct value of $x_{N-3}$ is $i$. Then,

   $$X_{N-2,i} = \arg\max_j \left\{ P_j^{x_{N-2},i} \right\}. \tag{35}$$

   This argmax computation has complexity $O(\log_2 D)$ with pair comparisons, so it requires the same time as the contraction of $Z^{N-1} - Z^{N-2}$, making them parallel.

3. For $n$ from $N-2$ to 1, defining $x_{-1} = 0$:

   - We contract $L^n$ with $Z^{n-1}$ to get $L^{n-1}$ in a BigUnit, computing each of its elements with a SmallUnit.
   - The contraction of $L^n$ with $V^{n-1}$ for every possible value of $x_{n-2}$ to obtain all the vectors $P^{x_{n-1},x_{n-2}}$.
   - Now, for each value of $x_{n-2}$, we save the position of the largest component, obtaining the vector $X_{n-1}$. This vector component $X_{n-1,i}$ has the correct values for $x_{n-1}$ if the correct value of $x_{n-2}$ is $i$.
   - We must perform these steps iteratively, so each iteration has a computational complexity of $O(\log_2 D)$ and a space complexity of $O(D^2)$. Repetition of $N$ times increases computational complexity to $O(N \log_2 D)$. The space complexity remains $O(ND + D^2)$, because we can overwrite the vectors $P$ at each step, and we only need to store the vectors $X$ of $D$ components.

4. We take $X_0$, which has only one component, because there is no previous variable, so $x_0 = X_{0,0}$. This step has computational complexity $O(1)$.

5. For each $n$ from 1 to $N-2$, we select $x_n = X_{n,x_{n-1}}$. Each iteration has computational complexity $O(1)$, so the total has complexity $O(N)$.

6. The last variable value is determined by brute force, which requires $O(\log_2 D)$ calculations.

We call *MeLoCoToN backtracking* to this last precomputation technique for the final backtracking evaluation of the solution. Taking into account all computational complexities and space complexities, the total computational complexity of the parallel algorithm is $O(N \log_2 D)$ and the space complexity is $O(ND + D^2)$ (in Tensor QUDO it is $O(ND^2)$). So, the runtime of this algorithm scales linearly with the number of variables and logarithmically with their dimension. Tab. I shows the different computational and space complexity for both algorithms.

| Algorithm | Computational Complexity | Space Complexity (QUDO\|T-QUDO) |
|---|---|---|
| Brute Force | $O(D^N)$ | $O(N)\|O(ND^2)$ |
| LCQPTNS (this work) | $O(N^2D^2)$ | $O(N+D^2)\|O(ND^2)$ |
| LCQPTNS with reuse (this work) | $O(ND^2)$ | $O(ND+D^2)\|O(ND^2)$ |
| Parallelized LCQPTNS (this work) | $O(N\log_2 D)$ | $O(ND+D^2)\|O(ND^2)$ |

TABLE I. Comparison of the different QUBO, QUDO and Tensor QUDO solvers for lineal chain with one-neighbor interaction for $N$ variables of values in the set $\{0, D-1\}$.

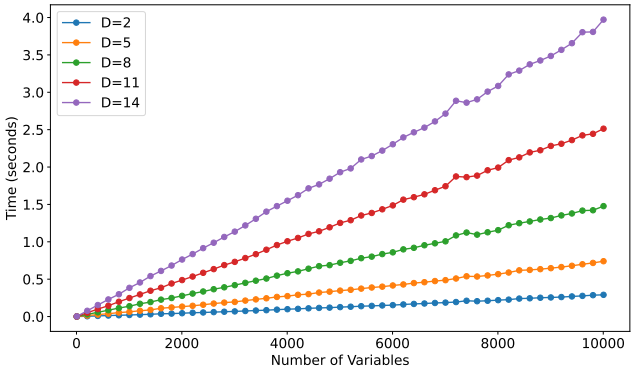


FIG. 4. Run time for QUDO solver algorithm in NumPy against number of variables of the problem.

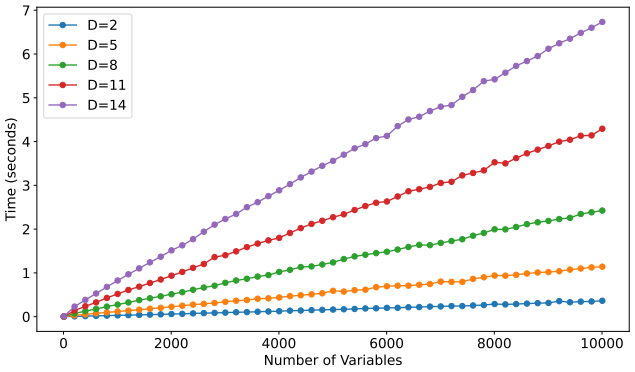


FIG. 5. Run time for QUDO solver algorithm in Decimal against number of variables of the problem

## IV. EXPERIMENTS

In this section, we introduce several experiments to analyze the performance of the proposed algorithms and compare it with the Google OR-TOOLS [30] and dimod [31] solvers. Tensor network algorithms are implemented in Python without parallelization, and all code is available in the GitHub repository. Every runtime experiment is performed three times and averaged to avoid fluctuations. In our implementation in NumPy, the value of $\tau$ for the new node increases to $\tau_n = \tau N/(N-n)$ to compensate for the reduction in the number of nodes. This considerably improves the quality of the results due to the finite precision of the numbers. We also normalize each tensor by its norm, and the intermediate computation tensors are also normalized. Instances are generated by uniform random numbers in the range $(-1, 1)$. We also normalize the $w$ matrices/tensors and the $d$ vectors by their norms. We also normalize the $Z$ tensors and, for the Decimal implementation, if an overflow happens, we rescale $\tau$ until it does not diverge. However, the overflow problem remains, so we also implement the algorithm with the Decimal library for the QUDO algorithm, performing the multiplications with lists. This library allows to increase the precision of the numbers stored. All experiments are performed in CPU, with an Intel(R) Core(TM) i7-14700HX 2.10 GHz and 16 GB RAM.

First, we test the scaling of the runtime with the number $N$ of variables. Figs. 4, 5 and 6 show the runtime for QUDO algorithms in Numpy and in Decimal, and for the Tensor QUDO algorithm, with the different number

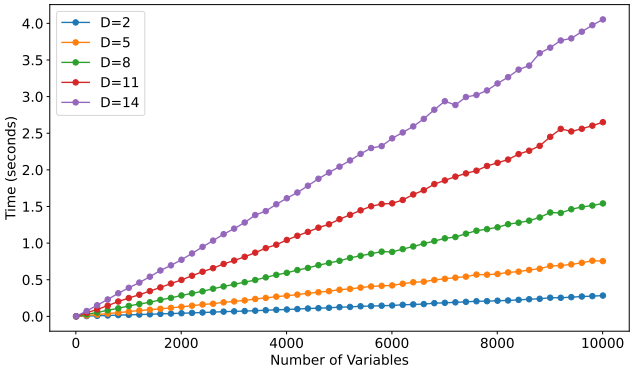


FIG. 6. Run time for T-QUDO solver algorithm in NumPy against number of variables of the problem.

of variables. Both figures show that the runtime of all algorithms scales linearly with the number of variables, as we described. The Decimal implementation has a higher runtime, as expected, due to the higher precision and more bits operations.

Now, we test the scaling of the runtime with the dimension $D$ of the variables. Figs. 7, 8 and 9 show the runtime for QUDO algorithms in Numpy and in Decimal, and for the Tensor QUDO algorithm, with the different dimension of the variables. Both figures show that the runtime of all algorithms scales quadratically with the dimension of the variables, as we described. As before, the Decimal implementation has a higher runtime.

Now, we compare the quality of the solutions with the

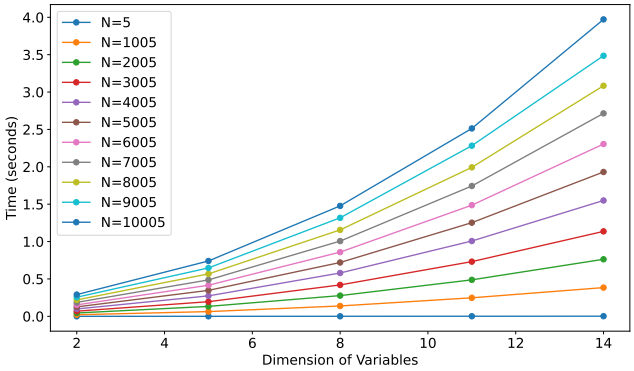


FIG. 7. Run time for QUDO solver algorithm in NumPy against the dimension of variables of the problem.

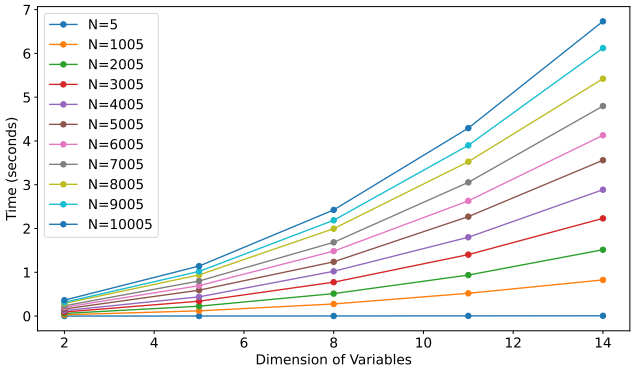


FIG. 8. Run time for QUDO solver algorithm in Decimal against the dimension of variables of the problem.

value of $\tau$ for QUBO problems. We take as the correct solution the best provided by the Google OR-TOOLS and dimod solvers. The cost ratio is defined as

$$\mathcal{R} = \frac{C(\vec{x}_{TN})}{C(\vec{x}_{opt})}, \tag{36}$$

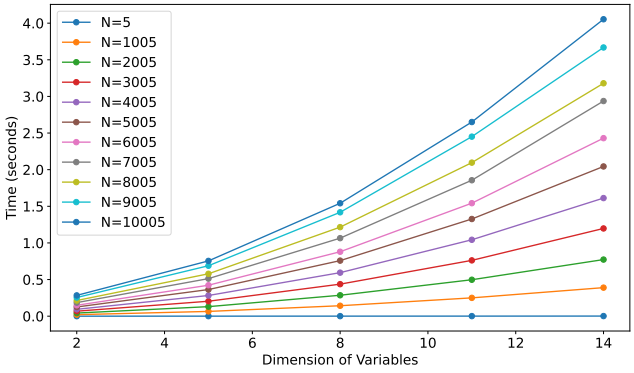


FIG. 9. Run time for T-QUDO solver algorithm in NumPy against the dimension of variables of the problem.

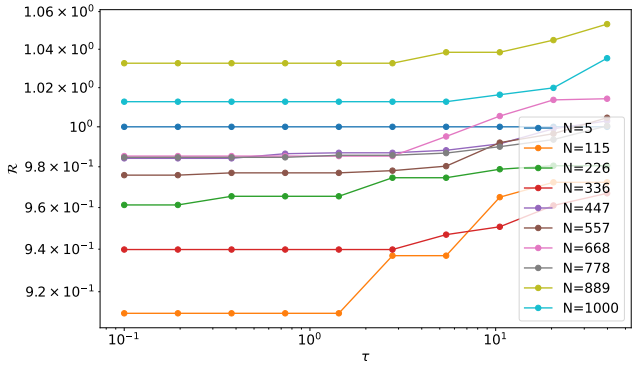


FIG. 10. Ratio between the cost of the solutions of QUBO tensor networks solver in Numpy and QUBO simulated annealing solver or ORTOOLS solver against $\tau$.

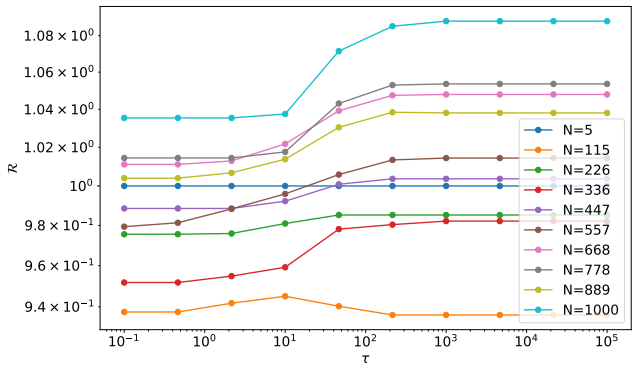


FIG. 11. Ratio between the cost of the solutions of QUBO tensor networks solver in Decimal and QUBO simulated annealing solver or ORTOOLS solver against $\tau$.

being $C(\vec{x}_{TN})$ our tensor network solution. We perform the comparison for $D = 2$ instances because it allows for a direct optimization by dimod's simulated annealing solver. For fair comparison, dimod and ORTOOLS are limited to a similar runtime to the tensor networks algorithms. Figs. 10, 11 show the cost ratio for QUDO algorithms in Numpy and in Decimal with the different values of $\tau$. We can see that, in general, there is an improvement in the ratio with the value of $\tau$, obtaining even better performance than dimod and ORTOOLS. We can see that, as we increase the number of variables, the tensor network results are better compared to the classical solvers. This shows the potential of this algorithm for larger instances.

## V. CONCLUSIONS

We have developed several algorithms in tensor networks to solve these lineal chain one-neighbor interaction quadratic problems efficiently, and provide the exact explicit equation that solves them. Moreover, we have im-

plemented and tested them with other well-known classical algorithms and observed an improvement in the quality of the results. However, open questions remain. The first question is the minimal finite value of $\tau$ that results in the correct optimal solution. This depends on the dimensions and properties of the problem to solve and could be a future line of research. For example, a useful technique could be the iterative normalization described in [29]. Another possible line will be the efficient resolution of more general QUBO, QUDO and Tensor QUDO problems with this methodology and its application to various industrial problems. In addition, it is possible to explore how to improve the computational complexity of the algorithm by inserting techniques such as quantization of the possible values of the tensor elements or taking advantage of the sparsity of the tensors. Finally, it is also interesting to test efficient implementations of the parallelized version of the algorithm we provided.

## ACKNOWLEDGMENTS

The research leading to this paper has received funding from the Q4Real project (Quantum Computing for Real Industries), HAZITEK 2022, no. ZE-2022/00033.

[1] F. Glover, G. Kochenberger, and Y. Du, A tutorial on formulating and using qubo models (2019), arXiv:1811.11538 [cs.DS].

[2] J. F. A. Sales and R. A. P. Araos, Adiabatic quantum computing for logistic transport optimization (2023), arXiv:2301.07691 [quant-ph].

[3] S. V. Romero, E. Osaba, E. Villar-Rodriguez, and A. Asla, Solving logistic-oriented bin packing problems through a hybrid quantum-classical approach, in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)* (2023) pp. 2239–2245.

[4] T. Matsumori, M. Taki, and T. Kadowaki, Application of qubo solver using black-box optimization to structural design for resonance avoidance, Scientific Reports **12**, 12143 (2022).

[5] T. Zaborniak, J. Giraldo, H. Müller, H. Jabbari, and U. Stege, A qubo model of the rna folding problem optimized by variational hybrid quantum annealing, in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2022) pp. 174–185.

[6] M. Mattesi, L. Asproni, C. Mattia, S. Tufano, G. Ranieri, D. Caputo, and D. Corbelletto, Diversifying investments and maximizing sharpe ratio: a novel qubo formulation (2024), arXiv:2302.12291 [quant-ph].

[7] A. Mata Ali, Introduction to qudo, tensor qudo and hobo formulations: Qudits, equivalences, knapsack problem, traveling salesman problem and combinatorial games.

[8] H. Yasuoka, Computational complexity of quadratic unconstrained binary optimization (2022), arXiv:2109.10048 [cs.CC].

[9] E. Çela and A. P. Punnen, Complexity and polynomially solvable special cases of qubo, in *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications*, edited by A. P. Punnen (Springer International Publishing, Cham, 2022) pp. 57–95.

[10] K. Allemand, K. Fukuda, T. M. Liebling, and E. Steiner, A polynomial case of unconstrained zero-one quadratic optimization, Mathematical Programming **91**, 49 (2001).

[11] K. Nakano, D. Takafuji, Y. Ito, T. Yazane, J. Yano, S. Ozaki, R. Katsuki, and R. Mori, Diverse adaptive bulk search: a framework for solving qubo problems on multiple gpus, in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2023) pp. 314–325.

[12] D. Aharonov, I. Arad, and S. Irani, Efficient algorithm for approximating one-dimensional ground states, Phys. Rev. A **82**, 012315 (2010).

[13] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, Physics-inspired optimization for quadratic unconstrained problems using a digital annealer, Frontiers in Physics **7**, 10.3389/fphy.2019.00048 (2019).

[14] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm (2014), arXiv:1411.4028 [quant-ph].

[15] Ákos Nagy, J. Park, C. Zhang, A. Acharya, and A. Khan, Fixed-point grover adaptive search for qubo problems (2023), arXiv:2311.05592 [quant-ph].

[16] I. P. Delgado, B. G. Markaida, A. M. F. De Leceta, and J. A. O. Uriarte, Quantum hobbit routing: Annealer implementation of generalized travelling salesperson problem, in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)* (2022) pp. 923–929.

[17] S. Pramanik and M. G. Chandra, Quantum-assisted graph clustering and quadratic unconstrained d-ary optimisation (2021), arXiv:2004.02608 [quant-ph].

[18] Y. Yan, Z. Du, J. Chen, and X. Ma, Limitations of noisy quantum devices in computational and entangling power (2023), arXiv:2306.02836 [quant-ph].

[19] J. Biamonte and V. Bergholm, Tensor networks in a nutshell (2017), arXiv:1708.00006 [quant-ph].

[20] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of Physics **349**, 117–158 (2014).

[21] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, Tensorizing neural networks (2015), arXiv:1509.06569 [cs.LG].

[22] A. Tomut, S. S. Jahromi, A. Sarkar, U. Kurt, S. Singh, F. Ishtiaq, C. Muñoz, P. S. Bajaj, A. Elborady, G. del Bimbo, M. Alizadeh, D. Montero, P. Martin-Ramiro, M. Ibrahim, O. T. Alaoui, J. Malcolm, S. Mugel, and R. Orus, Compactifai: Extreme compression of large language models using quantum-inspired tensor networks (2024), arXiv:2401.14109 [cs.CL].

[23] P. Seitz, I. Medina, E. Cruz, Q. Huang, and C. B. Mendl, Simulating quantum circuits using tree tensor networks, Quantum **7**, 964 (2023).

[24] K. Sozykin, A. Chertkov, R. Schutski, A.-H. Phan, A. Cichocki, and I. Oseledets, Ttopt: A maxi-

mum volume quantized tensor train-based optimization and its application to reinforcement learning (2022), arXiv:2205.00293 [cs.LG].

[25] T. Hao, X. Huang, C. Jia, and C. Peng, A quantum-inspired tensor network algorithm for constrained combinatorial optimization problems, Frontiers in Physics **10**, 10.3389/fphy.2022.906590 (2022).

[26] Y. Fang and A. S. Lele, Solving quadratic unconstrained binary optimization with collaborative spiking neural networks, in *2022 IEEE International Conference on Rebooting Computing (ICRC)* (2022) pp. 84–88.

[27] N. Nakatani, Matrix product states and density matrix renormalization group algorithm, in *Reference Module in Chemistry, Molecular Sciences and Chemical Engineering* (Elsevier, 2018).

[28] A. M. Ali, Explicit solution equation for every combinatorial problem via tensor networks: Melocoton (2025), arXiv:2502.05981 [cs.ET].

[29] A. M. Ali, I. P. Delgado, M. R. Roura, and A. M. F. de Leceta, Efficient finite initialization for tensorized neural networks (2023), arXiv:2309.06577 [cs.LG].

[30] L. Perron and V. Furnon, Or-tools.

[31] D-Wave Systems Inc., dimod.