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

Alejandro Mata Ali[1⋆], Iñigo Perez Delgado[2†], Marina Ristol Roura[2‡] and Aitor Moreno Fdez. de Leceta[3○]

1 i3B Ibermatica, Quantum Development Department, Paseo Mikeletegi 5, 20009 Donostia, Spain
2 i3B Ibermatica, Parque Tecnológico de Bizkaia, Ibaizabal Bidea, Edif. 501-A, 48160 Derio, Spain
3 i3B Ibermatica, Unidad de Inteligencia Artificial, Avenida de los Huetos, Edificio Azucarera, 01010 Vitoria, Spain

⋆ alejandro.mata.ali@gmail.com , † iperezde@ayesa.com , ‡ mristol@ayesa.com , ○ aitormoreno@lksnext.com

Abstract

This work presents a quantum-inspired tensor network algorithm for solving tridiagonal Quadratic Unconstrained Binary Optimization (QUBO) problems and quadratic unconstrained discrete optimization (QUDO) problems in linear time with the number of variables. This algorithm also can 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, the degenerate case is addressed, and the polynomial complexity of the algorithm is evaluated, also providing a parallelized version. These algorithms are implemented and tested with other well-known classical algorithms, and an improvement in the quality of the results was observed. The performance of the proposed algorithms is compared with the Google OR-TOOLS and dimod solvers, improving their results.

Received Date
Accepted Date
Published Date

Contents

1 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 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 classi-

cal 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.

This paper explores an efficient tensor network algorithm to solve general QUBO, QUDO, and Tensor QUDO problems with one-neighbor interactions in a lineal chain. It is called 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. Its applicability to degenerate cases is analyzed and its computational complexity is calculated. 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, Sec. 2 describes the problems to solve and presents a brief background on the state-of-the-art approach to solving them. Then, Sec. 3 introduces the three novel algorithms. Finally, Sec. 4 performs several experiments to analyze its performance.

All code is publicly available on the GitHub repository
https://github.com/DOKOS-TAYOS/Lineal_chain_QUBO_QUDO_TensorQUDO
and the reader can check the Streamlit of the algorithm in
https://lineal-chain-qubo-qudo-with-tensor-networks.streamlit.app/

2 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, the problem is to search 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

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

which implies that w is a tridiagonal matrix

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

In the Tensor QUDO case, the analogous problem would be

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

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 χ the required bond dimension, and could be extremely low for large systems. Additionally, this algorithm is not designed for Tensor QUDO problems.

3 Tensor Network algorithms

This section introduces the new tensor network algorithms for solving QUBO, QUDO and Tensor QUDO problems in a lineal chain with one-neighbor interaction efficiently. First, Sec. 3.1 introduces the QUBO algorithm, as a first and simpler version. Secondly, Sec. 3.2 introduces how to generalize it for QUDO problems. Third, Sec. 3.3 generalizes it for Tensor QUDO problems. Then, Sec. 3.4 discusses how to optimize the tracing method, and Sec. 3.5 addresses the degenerate case. Finally, Sec. 3.6 analyzes the computational complexity of the execution of algorithms.

3.1 Tridiagonal QUBO tensor network solver

First, a solver for the tridiagonal QUBO problem is created, and then it is generalized to the QUDO problem. A modified version of the method [25] is used and 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, an approximate optimal solution can be determined in $O(N)$ time, which is the optimal in the limit as $\tau \to \infty$.*

For this, the basics of the algorithm is explained 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.

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, this tensor represents a 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

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

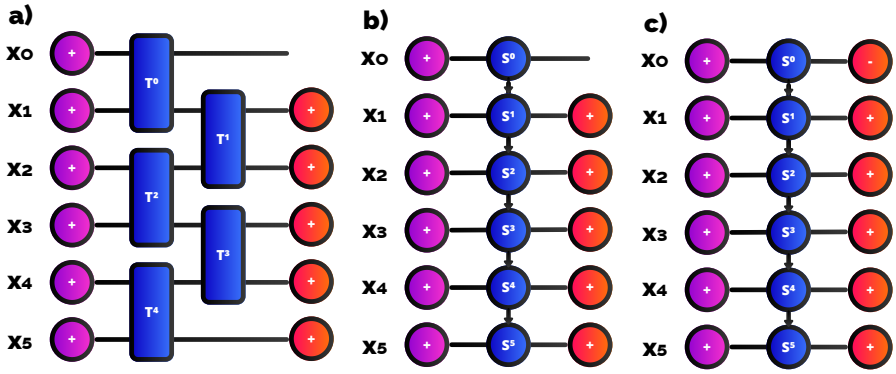

Figure 1: Tensor networks solving the tridiagonal QUBO problem and the one-neighbor QUDO problem. a) Bilocal version, b) Matrix Product Operator (MPO) version, c) Matrix Product Operator (MPO) version for explicit equation.

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

$$\begin{aligned} &T^n_{2\times2\times2\times2}, \\ &\mu = i, \nu = j, \\ T^n_{ij\mu\nu} &= e^{-\tau(w_{n,n+1}ij + w_{n,n}i)}, \end{aligned} \tag{10}$$

$$\begin{aligned} &T^{N-2}_{2\times2\times2\times2}, \\ &\mu = i, \nu = j, \\ T^{N-2}_{ij\mu\nu} &= e^{-\tau(w_{N-2,N-1}ij + w_{N-2,N-2}i + w_{N-1,N-1}j)}, \end{aligned} \tag{11}$$

where τ 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.

Since the resulting tensor has 2^N components, we cannot simply look at which component has the largest amplitude. Its information must be extracted in a more efficient way. The amplitude of the lowest-cost combination is assumed to be sufficiently larger than the amplitudes of the other combinations. This is a reasonable assumption because if τ is increased, 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 the value of x_0 associated with the optimal combination is obtained. 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. This operation is called *Half Partial Trace* with x_0 free, which returns a vector P^{x_0} with components.

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

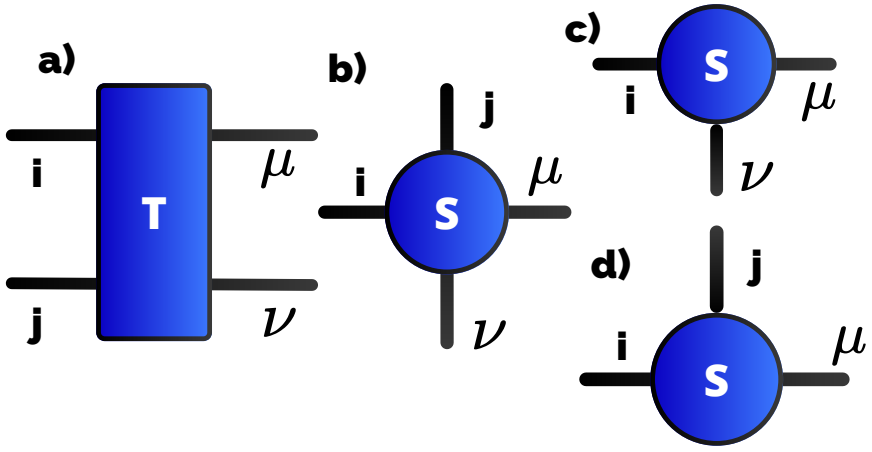

Figure 2: Tensor indices. a) T tensor, b) First S tensor, c) Intermediate S tensor, d) Final S tensor.

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 τ, 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 the resulting vector is multiplied 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 at the 0-th position for a τ value. This is represented in Fig. 1 c for six variables.

The contraction of the tensor network can be optimized by defining it as shown in Fig. 1 b, where the T-tensors are replaced 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. S^n is 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

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

$$S^n_{2\times2\times2\times2},\quad \mu=\nu=i,\quad S^n_{ij\mu\nu}=e^{-\tau(w_{n-1,n}ij+w_{n,n}i)}, \tag{16}$$

$$S^{N-1}_{2\times2\times2},\quad \mu=i,\quad S^{N-1}_{ij\mu}=e^{-\tau(w_{N-2,N-1}ij+w_{N-1,N-1}i)}. \tag{17}$$

With the corresponding tensor network, the variable x_0 is determined and then these steps are followed 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}, the classical comparison is used:
   $$x_{N-1}=H(-(w_{N-1,N-1}+w_{N-2,N-1}x_{N-2})). \tag{19}$$

The progressive reduction of the represented variables is due to the fact that if the solution is already known, the rest of the unknown system can be considered by introducing the known information into it. 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, the variables reduction is omitted to simplify the explanation. With a tensor network of layers '+' and S, the n-th variable can be extracted by connecting the '+' nodes for all variables, but the n-th one, which will be connected to a minus vector. The resulting tensor network is called $\Omega_n(\tau)$. Knowing that this algorithm is exact in the limit if there is no degeneracy, the solution can be expressed as

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

This equation is exact because all the arguments previously shown and explicit because the solution does not depend on the own wanted solution. 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.

3.2 Tridiagonal QUDO problem

In the QUDO problem, the same tensor network structure can be used. The only modification is to change the 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, an approximate optimal solution can be determined 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 used 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 ν.

The rest of the algorithm is the same until the value extraction. If this tensor network is contracted analogously to the QUBO case, a vector P^{x_0} of dimension D_0 can be obtained. From this, the optimal value of x_0 can be extracted 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, exactly the same process that has been explained for the QUBO case is performed. The last step is changed to a comparison that gives us the value of x_{N-1}, which will result in a lower cost based on the already known value of x_{N-2}.

To obtain the explicit equation, in the index of the variable to determine, a minus vector cannot simply connected because the dimensions do not fit. However, the value of x_n can be binarized into a set of binary variables $x_{n,m}$. This means that instead of connecting a minus vector, a *bit-selector vector* B is connected. 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.

3.3 One-neighbor Tensor QUDO problem

In this case, the focus is to solve the lineal chain one-neighbor Tensor QUDO problem using an extended version of the tridiagonal QUDO algorithm. Here, the only modification is 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, an approximate optimal solution can be determined 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.

3.4 Tracing optimization

One of the biggest problems that can arise is choosing a value of τ 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, τ can be chosen as a constant. A good practice to ensure this may be that the maximum cost that can be obtained 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 that, when the Half Partial Trace is performed, 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.

These phases are added 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. It is called the *Humbucker* method, inspired by the noise cancellation for guitar coils. To improve performance, a small random factor can be added to each combination phase.

In this case, the only modification needed is to change the definition of P^{x_0} to

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

$\gamma_{\vec{x}}$ being 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, the precision of the numbers used is exceeded, obtaining divergences in the trace vector. Related to this problem, there exists the possibility that by multiplying so many nodes, all 0 values will be obtained due to lack of precision in our numbers. To deal with this, it can be chosen to divide the components of the n initialization '+' nodes of step n so that the algorithm tries to make the sum of the components of the final trace vector near D_n. That is, the objective is to have a 1-norm

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

D_n is chosen as the target value to account for the possibility of having large differences in D between the problems and the variables. To do so, each initialization node '+' at step n multiplies 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 ξ_n is the factor that accounts for how much $\left(\frac{1}{\sigma_{n-1}}\right)^{\frac{1}{N-n}}$ would be if this normalization were not applied, with $\xi_0 = 1$.

By doing this, the states will be normalized 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].

3.5 Degenerate case

Until now, a non-degenerate case has been considered. However, in the degenerate case, where there is more than one optimal combination, there are more peaks of similar amplitude. They are not exactly equal, because of the contribution of residual states. In the phased method, the two peaks can have opposite phases, so that they cancel out and the optimum cannot be distinguished. For this reason, the recommendation is 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 there are two peaks with two different values of x_n, the algorithm chooses one of them and the rest of the obtained combination will be the one associated exactly with the x_n chosen, avoiding the degeneracy problem. In addition, if the other states of the degeneracy are wanted, the same process can be repeated again, but choosing the other high-amplitude component instead.

3.6 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 μ 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.

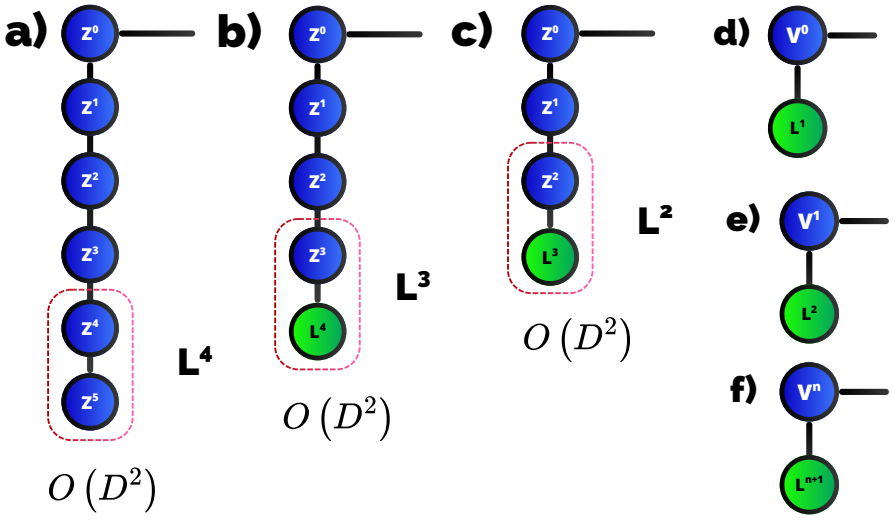

Figure 3: Contraction scheme for the efficient tensor network. This process is repeated $N-1$ times to obtain a component. a) Partial trace of the last variable. b) Partial trace of the penultimate variable. c) Contraction of the last variable with the penultimate one. d)

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

$$\begin{aligned} &Z^0_{D_0\times D_0},\\ &v=i,\\ &Z^0_{iv}=e^{-\tau w_{0,0,i,i}}, \end{aligned} \tag{32}$$

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

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

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, the computational and space complexity are $O(ND^2)$ for the creation and storage of tensors. However, if they are created just before using them, only one matrix must be stored 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. It has to be repeated $N-1$ 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 there are M degenerate solutions, to obtain all of them, the algorithm only has to apply the process M times, so there is a complexity of $O(MN^2D^2)$.

However, execution time can be improved by reusing intermediate computations. To do so, when calculating the result of the first variable, the intermediate tensors L^n that are shown

in Fig. 3 b and c are stored. In this way, since the algorithm only needs to change the n-th node for a new V^n tensor, as was explained for the iterative method, the rest of the tensor network, stored in the tensor L^{n+1} will be the same. Therefore, the algorithm will only need to multiply the new V^n tensor by the tensor L^{n+1} that had been stored from the first iteration.

Since each of these multiplications has a cost $O(D)$, because the matrix is diagonal, and it is repeated $O(N)$ times, the total cost of determining the variables from 1 to $N-2$ will be $O(ND)$. Since the cost to determine the first is $O(ND^2)$, there is 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 the Tensor QUDO the space complexity is $O(ND^2)$. In the QUBO case, the computational and space complexity is $O(N)$. If there are M degenerate solutions, to obtain all of them, the algorithm will only have to apply the process M times, so there is 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, the main assumption is that there are enough units for computation that can compute in parallel, so the parallel runtime with infinite processors is analyzed. From this point on, computational complexity refers to the main component of this parallel runtime. The intention is to determine the vector after the Half Partial Trace without knowing the previous variables results. That is, these vectors are computed 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. It will be better understood with the explicit algorithm. *BigUnit* is defined 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. Each Z tensor is created. 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, they are only generated just before contracting them.

2. At the same time, in $D+1$ different BigUnits, the algorithm computes:

   - Z^{N-1} is contracted 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}, the position of the largest component, obtaining the vector X_{N-2} is saved. 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,

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

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 1: 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 $\{0, D-1\}$.

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 $x_{-1} = 0$:

   - L^n is contracted 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}, the position of the largest component is saved, 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.
   - These steps must be performed 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 the vectors P can be overwritten at each step, and only the vectors X of D components need to be stored.

4. X_0 is taken, 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 $N-2$, $x_n = X_{n,x_{n-1}}$ is selected. 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.

It is called *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. 1 shows the different computational and space complexity for both algorithms.

4 Experiments

In this section, several experiments are introduced 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 τ for the new

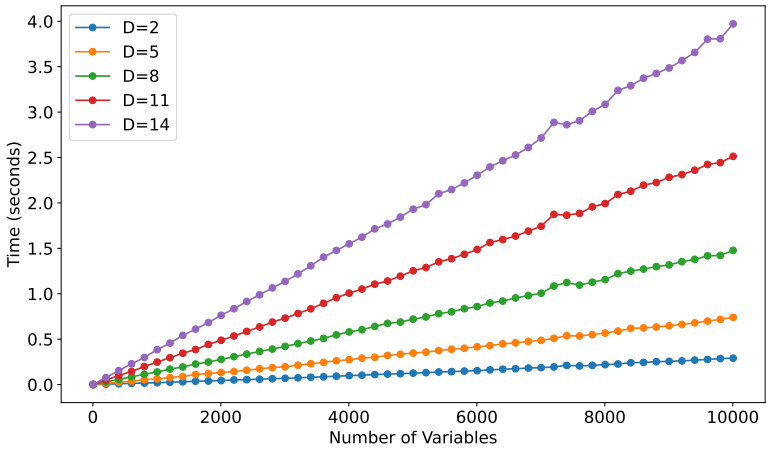

Figure 4: Run time for QUDO solver algorithm in NumPy against number of variables of the problem.

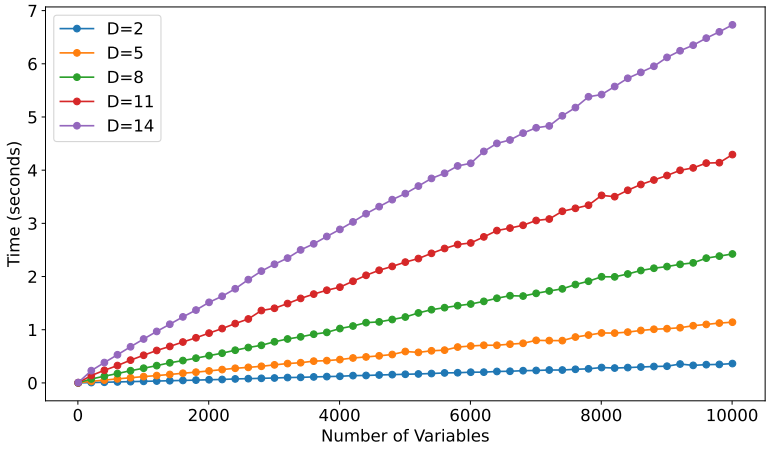

Figure 5: Run time for QUDO solver algorithm in Decimal against number of variables of the problem

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. Each tensor is normalized by its norm, and the intermediate computation tensors are also normalized. Instances are generated by uniform random numbers in the range $(-1, 1)$. The w matrices/tensors, the d vectors and the Z tensors are also normalized by their norms. However, the overflow problem remains, so the algorithm is also implemented with the Decimal library for the QUDO algorithm, performing the multiplications with lists. This library allows to increase the precision of the numbers stored. For the Decimal implementation, if an overflow happens, τ is rescaled until it does not diverge. All experiments are performed in CPU, with an Intel(R) Core(TM) i7-14700HX 2.10 GHz and 16 GB RAM.

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

Now, the scaling of the runtime with the dimension D of the variables is tested. Figs. 7, 8

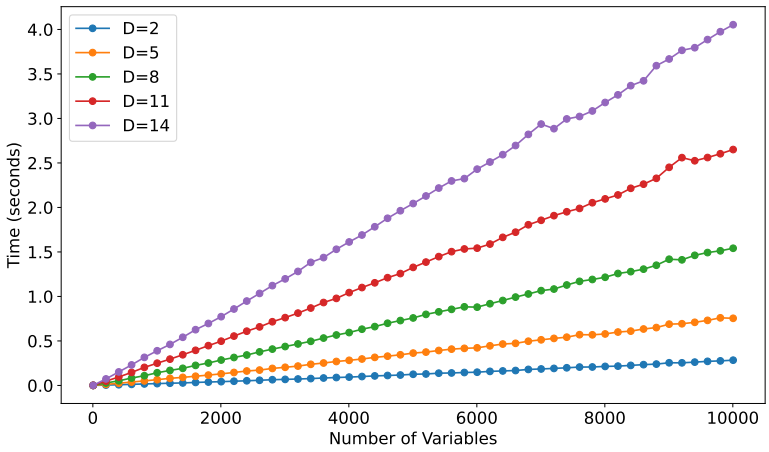

Figure 6: Run time for T-QUDO solver algorithm in NumPy against number of variables of the problem.

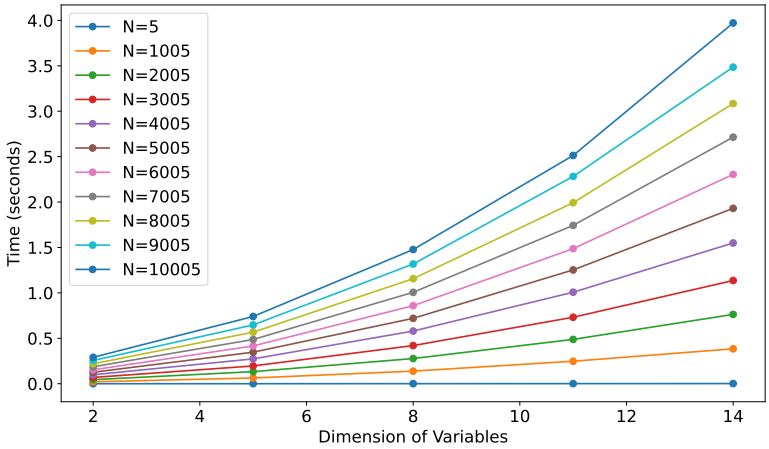

Figure 7: Run time for QUDO solver algorithm in NumPy against the dimension of variables of the problem.

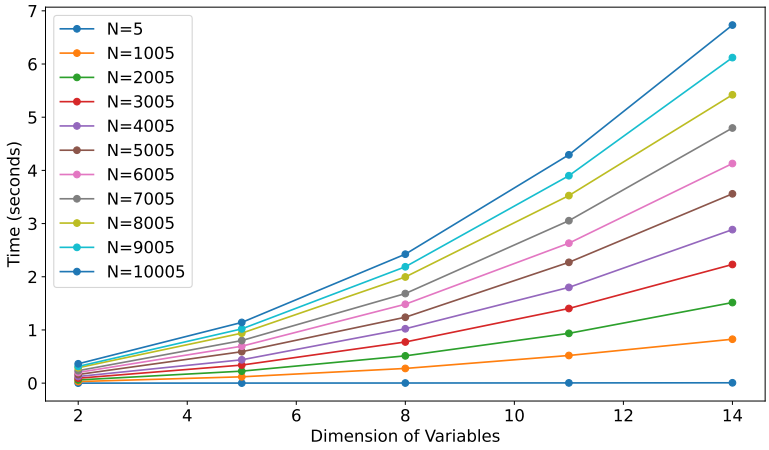

Figure 8: Run time for QUDO solver algorithm in Decimal against the dimension of variables of the problem.

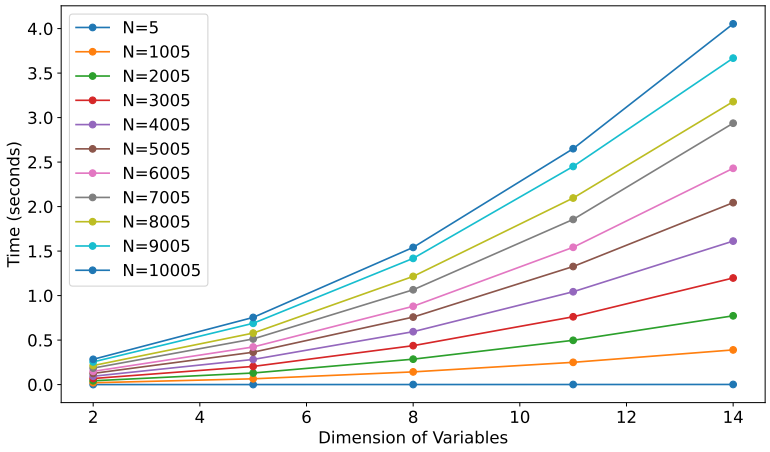

Figure 9: Run time for T-QUDO solver algorithm in NumPy against the dimension of variables of the problem.

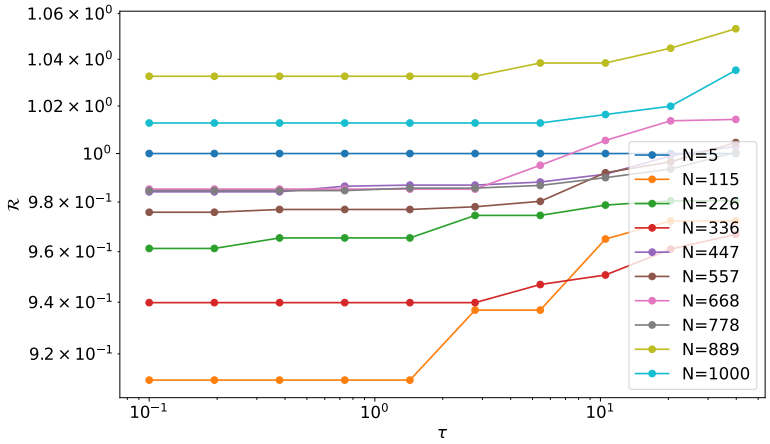

Figure 10: Ratio between the cost of the solutions of QUBO tensor networks solver in Numpy and QUBO simulated annealing solver or ORTOOLS solver against τ.

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 described. As before, the Decimal implementation has a higher runtime.

Now, the quality of the solutions with the value of τ is compared for QUBO problems. The best solution provided by the Google OR-TOOLS and dimod solvers is taken as the correct one. The cost ratio is defined as

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

being $C(\vec{x}_{TN})$ our tensor network solution. The comparison for $D = 2$ instances is performed 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 τ. In general, there is an improvement in the ratio with the value of τ, which is even better than dimod and ORTOOLS. As the number of variables increases, the results of the tensor network are better compared to the classical solvers. This shows the

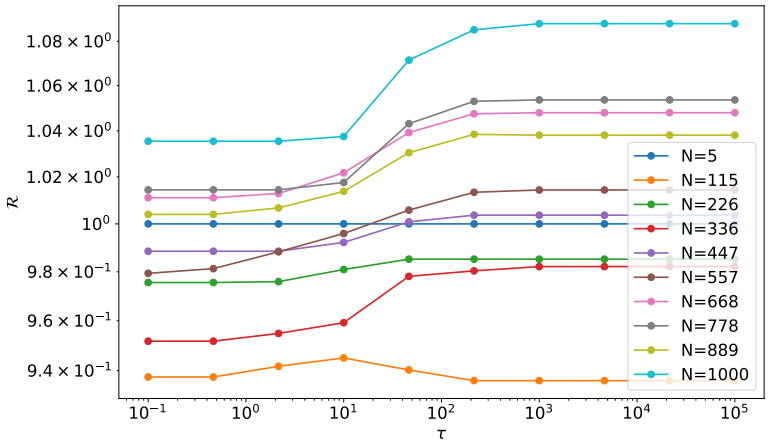

Figure 11: Ratio between the cost of the solutions of QUBO tensor networks solver in Decimal and QUBO simulated annealing solver or ORTOOLS solver against τ.

potential of this algorithm for larger instances.

5 Conclusions

This work 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, they have been implemented and tested 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 τ 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 or even solving ground state determination for hamiltonians of Ising type. 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 provided.

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

References

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

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

[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), pp. 2239–2245, doi:10.1109/ITSC57777.2023.10422581 (2023).

[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**(1), 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)*, pp. 174–185, doi:10.1109/QCE53715.2022.00037 (2022).

[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), 2302.12291.

[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), 2109.10048.

[9] E. Çela and A. P. Punnen, *Complexity and Polynomially Solvable Special Cases of QUBO*, pp. 57–95, Springer International Publishing, Cham, ISBN 978-3-031-04520-2, doi:10.1007/978-3-031-04520-2_3 (2022).

[10] K. Allemand, K. Fukuda, T. M. Liebling and E. Steiner, *A polynomial case of unconstrained zero-one quadratic optimization*, Mathematical Programming **91**(1), 49 (2001), doi:10.1007/s101070100233.

[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)*, pp. 314–325, doi:10.1109/IPDPSW59300.2023.00060 (2023).

[12] D. Aharonov, I. Arad and S. Irani, *Efficient algorithm for approximating one-dimensional ground states*, Phys. Rev. A **82**, 012315 (2010), doi:10.1103/PhysRevA.82.012315.

[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** (2019), doi:10.3389/fphy.2019.00048.

[14] E. Farhi, J. Goldstone and S. Gutmann, *A quantum approximate optimization algorithm* (2014), 1411.4028.

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

[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)*, pp. 923–929, doi:10.1109/SSCI51031.2022.10022127 (2022).

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

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

[19] J. Biamonte and V. Bergholm, *Tensor networks in a nutshell* (2017), 1708.00006.

[20] R. Orús, *A practical introduction to tensor networks: Matrix product states and projected entangled pair states*, Annals of Physics **349**, 117–158 (2014), doi:10.1016/j.aop.2014.06.013.

[21] A. Novikov, D. Podoprikhin, A. Osokin and D. Vetrov, *Tensorizing neural networks* (2015), 1509.06569.

[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 *et al.*, *Compactifai: Extreme compression of large language models using quantum-inspired tensor networks* (2024), 2401.14109.

[23] P. Seitz, I. Medina, E. Cruz, Q. Huang and C. B. Mendl, *Simulating quantum circuits using tree tensor networks*, Quantum **7**, 964 (2023), doi:10.22331/q-2023-03-30-964.

[24] K. Sozykin, A. Chertkov, R. Schutski, A.-H. Phan, A. Cichocki and I. Oseledets, *Ttopt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning* (2022), 2205.00293.

[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** (2022), doi:10.3389/fphy.2022.906590.

[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)*, pp. 84–88, doi:10.1109/ICRC57508.2022.00021 (2022).

[27] N. Nakatani, *Matrix product states and density matrix renormalization group algorithm*, In *Reference Module in Chemistry, Molecular Sciences and Chemical Engineering*. Elsevier, ISBN 978-0-12-409547-2, doi:https://doi.org/10.1016/B978-0-12-409547-2.11473-8 (2018).

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

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

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

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