Knapsack and Shortest Path Problems Generalizations from a Quantum-Inspired Tensor Network Perspective

Sergio Muñiz Subiñas[1][a], Jorge Martínez Martín[1][b], Alejandro Mata Ali[1][c], Javier Sedano[1][d] and Ángel Miguel García-Vico[2][e]

[1]*Instituto Tecnológico de Castilla y León, Burgos, Spain*

[2]*Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Jaén, 23071 Jaén, Spain*

Keywords: Quantum Computing, Tensor Networks, Combinatorial Optimization, Quantum-Inspired.

Abstract: In this paper, we present two tensor network quantum-inspired algorithms to solve the knapsack and the shortest path problems, and enables to solve some of its variations. These methods provide an exact equation which returns the optimal solution of the problems. As in other tensor network algorithms for combinatorial optimization problems, the method is based on imaginary time evolution and the implementation of restrictions in the tensor network. In addition, we introduce the use of symmetries and the reutilization of intermediate calculations, reducing the computational complexity for both problems. To show the efficiency of our implementations, we carry out some performance experiments and compare the results with those obtained by other classical algorithms.

1 INTRODUCTION

Combinatorial optimization has become a wide field of study due to the large number of industrial and academic applications in different disciplines, for instance, engineering or logistics. Some of the most prominent and extensively studied problems are route optimization (traveling salesman problem (Laporte, 1992; Korte and Vygen, 2008) or the shortest path (Dijkstra, 1959; Hart et al., 1968)), task scheduling (job-shop scheduling problem (Lenstra, 1992; Dauzère-Pérès et al., 2024)) and constrained optimization (knapsack (Mathews, 1896; Bateni et al., 2018) or bin packing (Korte and Vygen, 2000; Egor et al., 2022)). The fact that most of these problems are NP-hard makes exact resolution methods computationally unaffordable. With the objective of solving these problems, it is common to draw upon approximations, generally obtaining a suboptimal result. However, in many industrial cases, the focus is on finding a solution that works well enough in a reasonable time, rather than looking for the best possible outcome. Some of the most used classical approximation techniques are greedy approaches (Gutin and Yeo, 2007), reinforcement learning (Wang and Dong, 2024), genetic algorithms (Pilcher, 2023) or simulated annealing (Katangur et al., 2002).

Meanwhile, due to the development of quantum technologies, new ways of approaching combinatorial optimization problems have been proposed. Among other applications, it is possible to address the solution for quadratic unconstrained binary optimization (QUBO) (Glover et al., 2019) problems with quantum algorithms such as the quantum approximate optimization algorithm (QAOA) (Farhi et al., 2014) or the variational quantum eigensolver (VQE) (Peruzzo et al., 2014). Likewise, Constrained Quadratic Models (CQM) and Binary Quadratic Models (BQM) can be solved with quantum annealers (Benson et al., 2023). With the current status of quantum computers hardware, where qubits are noisy and unreliable, no improvement in the state-of-the-art has been achieved. Nevertheless, due to the strong theoretical basis of these algorithms, researchers have developed quantum-inspired strategies, with tensor networks (TN) emerging as one of the most populars (Biamonte and Bergholm, 2017; Biamonte, 2020). This discipline consists in taking advantage of some prop-

[a] https://orcid.org/0009-0008-7590-0149
[b] https://orcid.org/0009-0004-9336-3165
[c] https://orcid.org/0009-0006-7289-8827
[d] https://orcid.org/0000-0002-4191-8438
[e] https://orcid.org/0000-0003-1583-2128

Subiñas, S. M., Martín, J. M., Ali, A. M., Sedano, J., García-Vico and Á. M.
Knapsack and Shortest Path Problems Generalizations from a Quantum-Inspired Tensor Network Perspective.
DOI: 10.5220/0013489000004525
In *Proceedings of the 1st International Conference on Quantum Software (IQSOFT 2025)*, pages 81-88
ISBN: 978-989-758-761-0

erties of tensor algebra to compress information or perform calculations in a more efficient way. Moreover, by applying tensor decomposition operations such as singular value decomposition (SVD) (Golub and Kahan, 1965) in a truncated way, it is possible to make approximations losing the least relevant information. Numerous studies show the computational advantages of tensor networks in a wide variety of applications, for instance, in the simulation of quantum systems (Biamonte, 2020), compression of neuronal networks (Qing et al., 2024), anomaly detection (Wang et al., 2020; Ali et al., 2024a) or combinatorial optimization problems (Hao et al., 2022), which are the object of study of this paper.

In this paper, we present two tensor network algorithms to solve the knapsack and shortest path problems. Our method is based on previous works applied to traveling salesman problem (Ali et al., 2024b), QUBO problems (Ali et al., 2024d) and task scheduling (Ali et al., 2024c). The objective of this study is to present an analytical methodology for solving combinatorial optimization problems, rather than a computational advantage in such problems. Following the results in (Ali, 2025), it provides an exact equation that returns the optimal solution to these problems. We compare the results of our implementations against some classical algorithms.

2 KNAPSACK PROBLEM

Given a set of n different item classes, where the i-th class can be selected up to c_i times, each with an associated weight $w_i \in \mathbb{N}$ and value $v_i \in \mathbb{R}^+$, the knapsack problem consists in finding the configuration $\vec{x} = (x_0, .., x_{n-1})$ that maximizes

$$\begin{aligned} V(\vec{x}) &= \sum_{i=0}^{n-1} x_i v_i \\ \text{subject to } W(\vec{x}) &= \sum_{i=0}^{n-1} x_i w_i \leq Q, \\ x_i \in [0, c_i] \quad \forall i \in [0, n-1], \end{aligned} \tag{1}$$

being x_i the number of times the i-th class has been selected, $V(\vec{x})$ the total value of the items, $W(\vec{x})$ the total weight of the items in the knapsack and Q the maximum weight capacity. There are different variants of this problem, the most popular being the 0-1 knapsack, where each class can be selected only once, $x_i \in \{0,1\}$. These instances variants can be solved exactly with a time complexity of $O(nQ)$ and space complexity $O(nQ)$.

2.1 Tensor Network Algorithm

The idea of our algorithm is to take advantage of the imaginary time evolution to generate a state whose basis state with the greatest amplitude represents the optimal configuration. Due to the high number of possible combinations, we cannot know the amplitude of all combinations at once, and hence determine the highest probability state. Therefore, we employ a method that consists in determining each variable separately, by projecting the state into each subspace associated with the respective variable.

2.1.1 Quantum-Inspired Tensor Network

Our knapsack problem algorithm is based on tensor algebra and some properties of quantum computing such as superposition or projected measurements. Thus, we have developed a quantum-inspired tensor network that incorporates the theoretical advantages of quantum computing without being limited by its constraints. The tensor network (see Fig. 2 a) can be interpreted as a 'quantum circuit' that can be simulated in four steps, each one corresponding to one of the layers in the figure.

The system starts in a uniform superposition of n variables, each one encoded in the basis states of a qudit of dimension $c_i + 1$. This first step corresponds to the first layer of tensors '+' in Fig. 2 a, and the mathematical representation of the resulting state is

$$|\psi_0\rangle = \sum_{\vec{x}} |\vec{x}\rangle = \bigotimes_{i=0}^{n-1} \sum_{x_i=0}^{c_i} |x_i\rangle. \tag{2}$$

The next step consists of applying an imaginary time evolution layer to the superposition that amplifies the basis states with a higher value. We implement this with an operator $\mathcal{S}$, which assigns to each state $|\vec{x}\rangle$ an exponential amplitude corresponding to its associated value $V(\vec{x})$. The parameter τ scales the costs so that the amplitudes differ exponentially from each other. The state after applying this layer is

$$|\psi_1\rangle = \mathcal{S}|\psi_0\rangle = \sum_{\vec{x}} e^{\tau V(\vec{x})} |\vec{x}\rangle. \tag{3}$$

In order to eliminate incompatible configurations, we apply a third layer that imposes the maximum capacity constraint,

$$\begin{aligned} |\psi_2\rangle = \mathcal{R}(Q)|\psi_1\rangle &= \sum_{\vec{x}} e^{\tau V(\vec{x})} \mathcal{R}(Q)|\vec{x}\rangle \\ \mathcal{R}(Q)\,|\vec{x}\rangle = |\vec{x}\rangle \text{ if } W(\vec{x}) &\leq Q, \end{aligned} \tag{4}$$

where $\mathcal{R}(Q)$ is the operator that projects the state into the subspace that fulfills the maximum weight capacity Q. This is implemented by a layer of R^i tensors,

where the up index keeps track of the current weight loaded at the knapsack until that qudit, while the down index outputs the resultant weight after adding the new qudit state. If this weight exceeds Q, the amplitude of the state is multiplied by zero.

Once obtained a system that has performed the minimization and constraint, we need a method to extract the position of the largest element of the superposition without checking all the possible combinations. To do this, we have to take into account that, for a sufficiently large τ, there will be a very high amplitude peak in the state of the optimal solution. In order to achieve this, we apply a partial trace layer to the tensor network except for the first qudit. This leads to the state

$$|\psi_3^0\rangle = \sum_{\vec{z}} \langle z_1, \ldots, z_{n-1}|\psi_2\rangle = \sum_{\vec{x}|W(\vec{x})\leq Q} e^{\tau V(\vec{x})}|x_0\rangle. \tag{5}$$

It can be interpreted as the resulting state from projecting $|\psi_2\rangle$ into the Hilbert space $\mathcal{H}^{c_0+1}$ of the first qudit and it is equivalent as the quantum process of measuring the first qubit. The associated vector representation is:

$$|\psi_3^0\rangle = \begin{pmatrix} \langle 0+\cdots+|\psi_2\rangle \\ \langle 1+\cdots+|\psi_2\rangle \\ \vdots \\ \langle c_0+\cdots+|\psi_2\rangle \end{pmatrix}, \tag{6}$$

where each amplitude allows to determine the probability of $|\psi_3^0\rangle$ of being in their corresponding basis states $|x_0\rangle$.

To determine the first variable x_0, we contract the tensor network (see Fig. 2 a) to obtain $|\psi_3^0\rangle$, whose highest amplitude value position corresponds to the optimum value of the variable x_0. To compute the remaining variables, we use the same tensor network but, instead of leaving free the index associated to the first variable, we free the index we want to obtain and follow the same contraction process as in the previous step. With this, we obtain the $|\psi_3^m\rangle$ state, from which we determine the m-th variable. In each step, at the end of the line of the variables that we have already determined, we put a vector where all its elements are zero, except the element corresponding to the chosen value of the variable. This process is shown in Fig. 1.

2.1.2 Tensor Network Layout

Since it is possible to compress the information of the four layers into a single layer in a convenient way, we propose a direct implementation of the tensor network in a chain structure (see Fig. 2). We define the tensors following the index convention of Fig. 3.

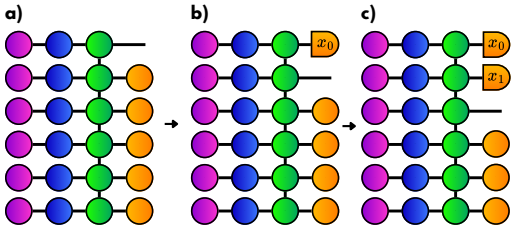

Figure 1: Graphic representation of the iterative method of the tensor network to obtain each variable.

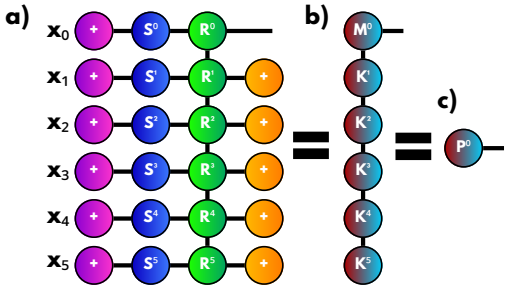

Figure 2: Tensor network solving the knapsack problem. a) Tensor network solving the knapsack problem composed of four column layers. From left to right: superposition ('+'), evolution (S), constraint (R) and tracing ('+'). b) Compressed version into a linear TN that implements all four layers simultaneously. c) Contracted tensor.

From the figure, we can see that there are three different types of tensors. The tensor M^0, with elements $M^0_{i\mu}$, is associated with the first qudit we want to measure. This tensor is the result of the contraction between the superposition node '+', the evolution node S^0 and the projector node R^0. It outputs information about the state of the qudit throughout the index i, and outputs the weight of the knapsack after adding the new items throughout the index μ. Therefore, the dimensions of the indexes i and μ are c'_0 and Q' respectively, being $Q' = Q+1$ and $c'_i = c_i + 1$ to streamline the notation. The non-zero elements of the

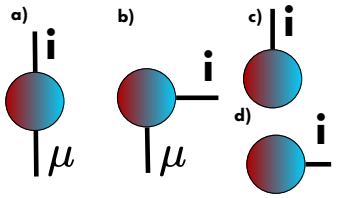

Figure 3: Nomenclature of the indexes of each tensor from the tensor chain, where i corresponds to a variable index and μ to a dependent index. a) A tensor from the middle of the tensor chain. b) The top tensor of the tensor chain. c) The bottom tensor of the tensor chain. d) The resulting tensor from the contraction of the tensor chain.

matrix $M^0_{c'_0 \times Q'}$ are defined as

$$\mu = iw_0, \quad M^0_{i\mu} = e^{\tau i v_0}. \tag{7}$$

The following $n-2$ tensors K^k belong to the same type, they have elements $K^k_{i\mu}$ and are associated with the k-th qudit. Each of these tensors receives information about the weight of the knapsack until the previous qudit throughout the index i, and passes the weight of the knapsack, including the weight added by this qudit, throughout the index μ. Therefore, the dimensions of the indexes i and μ are both Q'. The non-zero elements of $K^k_{Q' \times Q'}$ are defined as

$$y_k \in [0, c_k], \ \ \mu = i + y_k w_k, \ \ K^k_{i\mu} = e^{\tau y_k v_k}, \tag{8}$$

where y_k is the number of times we introduce the k-th element into the knapsack. This considers all possible additions of elements of that class that do not exceed the maximum weight.

Finally, the last tensor K^{n-1}, with elements K^{n-1}_i, is associated with the last qudit and receives from i the information of the knapsack of all the other qudits. The non-zero elements of $K^{n-1}_{Q'}$ are

$$y_i = \min\left(\frac{Q-i}{w_{n-1}}, c_{n-1}\right), \ \ K^{n-1}_i = e^{\tau y_i v_{n-1}}, \tag{9}$$

being y_i the maximum amount of elements of the class $n-1$ that could be added without exceeding the maximum weight capacity Q. Since in the event that one can have up to y_i positive value elements, the most optimal is always to have the maximum number.

It is important to note that by cutting the chain as the values of the variables are being set, the upper tensor of the chain will be different in each iteration, as it must include the information from the results obtained in the previous steps. Thus, during the iteration where the variable x_m is determined, our tensor network consists of the same $n-m-1$ last tensors and a new upper tensor $M^m_{c'_m \times Q'}$

$$\mu = iw_m + \sum_{k=0}^{m-1} x_k w_k, \ \ M^m_{i\mu} = e^{\tau i v_m}. \tag{10}$$

2.2 Contraction Scheme

In order to optimize the use of computational resources, we develop a contraction scheme that takes advantage of intermediate calculations. The idea is to store the resulting vectors from each step of the contraction process, while contracting the tensor chain bottom-up. This way, for each iteration of the algorithm, we only need to contract the new initial tensor with the corresponding stored intermediate tensor.

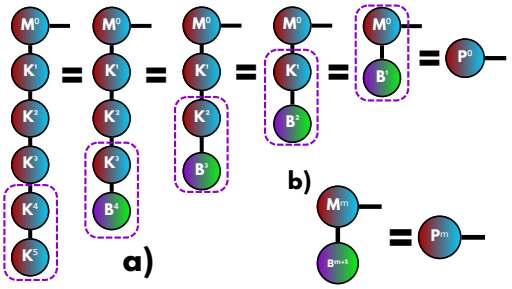

Figure 4: Contraction scheme with storage of intermediate tensors for later calculations.

The implementation of this recursive method can be seen in Fig. 4. In this process, the last node of the chain is contracted with the node directly above, resulting in a new node B^k, which will be stored in memory for later operations. We have to repeat this process until we have a single node P^0 which represents the $|\psi^0_3\rangle$ state. Once all the tensors are stored, we only need to contract the M^m matrices with their corresponding B^{m+1} tensor to obtain $|\psi^m_3\rangle$, contracting the whole tensor network of each iteration in one step.

2.3 Optimizations and Computational Complexity

As we have mentioned earlier, the tensor network algorithm that we implement needs to store about n $Q \times Q$ matrices. Therefore, to carry out the proposed contraction scheme, we need to store the result of n matrix-vector operations, in addition to performing each contraction between M^m and B^{m+1}.

Since each vector-matrix contraction has complexity $O(Q^2)$, the computational cost of the first step is $O(nQ^2)$, which is not optimal considering the structure of the matrices. It is possible to verify that the tensors K^k present sparsity; in particular, the number of nonzero elements in these matrices depends on c_k. All elements of K^k are zero except for c_k+1 diagonals, each one starts in column $c^i_k w_k$ where $c^i_k \in [0, c_k]$ and all its elements are equal to $e^{c^i_k \tau v_k}$. Therefore, the number of non-zero elements is $O(Qc_k)$.

Keeping this in mind and knowing that the computational complexity of multiplying a dense vector with a sparse matrix is $O(n_z)$, where n_z is the number of non-zero elements, the computational complexity of this step is $O(nQc)$, with c being the average c_k. As the second step also involves the multiplication of a sparse matrix and a dense vector, the computational complexity is likewise of the order $O(nQc)$. Thus, the total time complexity of the algorithm is $O(nQc)$, which is $O(nQ)$ in the 0-1 case.

For space complexity we have to take into account that each intermediate vector stored has $O(Q)$ elements. The matrices can be generated dynamically while contracting, so they require only $O(c)$ elements each, which is negligible compared to $O(Q)$. The total space complexity is $O(nQ)$.

In the case where intermediate calculations are not used, the contraction process must be carried out each time a variable is to be determined. This approach does not have the problem of storing n vectors of dimension Q, being the space complexity $O(Q)$. However, the computational complexity becomes $O(n^2Qc)$.

3 SHORTEST PATH PROBLEM

Given a graph G with $V \in \mathbb{N}$ vertices and a set E of edges, the shortest path problem, also called the single-pair shortest path problem, consist in finding a n-step path between two vertices

$$\vec{v} = (v_0, v_1, ..., v_{n-1}), \quad v_t \in [0, V], \tag{11}$$

such us the cost of the route is minimized, where v_t is the vertex associated to the t-th step. The cost of the route is given by

$$C(\vec{v}) = \sum_{t=0}^{n-2} E_{v_t, v_{t+1}}, \tag{12}$$

where $E_{ij} \in \mathbb{R}^+$ corresponds to the cost between the i-th vertex and j-th vertex. If two vertices are not connected, the cost between them is $E_{ij} = \infty$, and the cost between a node and itself is $E_{ii} = 0$.

The single-pair directional shortest path can be solved with a time complexity of $O((E+V)\log(V))$ and a space complexity $O(V)$ (Fredman and Tarjan, 1987).

3.1 Tensor Network Algorithm

The algorithm we propose in this paper is based on the tensor network in Fig. 5 a, relaying on the same principles explained in Ssec. 2.1.

3.1.1 Quantum-Inspired Tensor Network

The tensor network starts with a superposition layer equivalent to the state in Eq. 2. In this superposition, we apply an imaginary time evolution layer that damps the basis states with higher-cost routes. The state after applying this layer is

$$|\psi_1\rangle = \sum_{\vec{v}} e^{-\tau C(\vec{v})} |\vec{v}\rangle = \bigotimes_{t=0}^{n-2} \sum_{v_t=0}^{V-1} e^{-\tau E_{v_t, v_{t+1}}} |v_t\rangle. \tag{13}$$

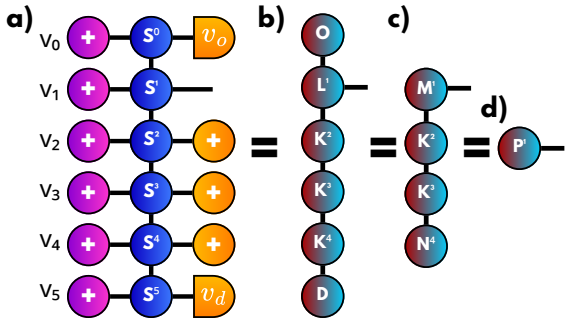

Figure 5: Tensor network solving the shortest path problem. a) Tensor network solving the shortest path problem composed of three column layers. From left to right: superposition ('+'), evolution (S) and tracing ('+'). b) Compressed version into a linear TN that implements all three layers simultaneously. c) Compressed version with the fixed nodes removed. d) Contracted tensor.

The last layer consists of a partial trace layer in each qudit except the second. For the first and last nodes, it is necessary to connect them to a node that represents the origin and destination nodes respectively, instead of the superposition node. This leads to the state

$$|\psi_2^1\rangle = \sum_{\vec{z}} \langle v_o, z_2, \ldots, v_d | \psi_1 \rangle = \sum_{\vec{v}} e^{-\tau C(\vec{v})} |v_1\rangle, \tag{14}$$

where v_o corresponds to the origin node and v_d to the destiny node.

3.1.2 Tensor Network Layout

As in the case of the knapsack, we propose a compressed formulation of the tensor network in a chain structure (see Fig. 5).

From the figure, we can see that there are three different types of tensors. The tensor L^1, with elements L^1_{ijk}, is associated with the qudit we want to measure. Given that the origin node is fixed, we remove it by manually adding the information to the next node resulting in the node M^1_{ij}. This tensor is the result of the contraction between the superposition node '+', the evolution node S^1 and the trace node. It outputs information about the state of the qudit throughout the index i, and outputs the selected node throughout the index j. Therefore, the dimensions of the indexes i and j are both V. The matrix $M^1_{V\times V}$ is defined as

$$M^1_{ij} = e^{-\tau(E_{oi} + E_{ij})}. \tag{15}$$

The following $n-3$ tensors belong to the same type, the tensor $K^k_{V\times V}$ which has elements K^k_{ij} is associated with the k-th qudit. It receives information about the previous visited node throughout the index i, and passes the new visited node throughout the index j. Therefore, the dimension of the indexes i and

j is V for both of them. $K^k_{V\times V}$ is defined as

$$K^k_{ij} = e^{-\tau E_{ij}}. \tag{16}$$

Finally, the last non-fixed tensor K^{n-2}_{ij} is associated with the second, since the last node corresponds to the destiny node. It receives from i the information of the previous node and, by fixing the last node to the destiny, we add the cost of the edge between the second last and the last node removing the index j. This result in the node N^{n-2}_V with the non-zero elements being

$$N^{n-2}_i = \sum_{j=0}^{V-1} e^{-\tau(E_{ij}+E_{jd})}. \tag{17}$$

It is important to note that by cutting the chain as the values of the variables are being set, the upper tensor of the chain will be different in each iteration as it must include the information from the results obtained in the previous steps. Thus, during the iteration where the variable v_m is determined, our tensor network consists of the same $n-m-1$ last tensors and a new upper tensor M^m_{ij}

$$M^m_{ij} = e^{-\tau(E_{pi}+E_{ij})}, \tag{18}$$

where p corresponds to the previous visited node. This tensor network follows the same contraction scheme as explained in Ssec. 2.2.

3.2 Optimizations and Computational Complexity

As mentioned before, the tensor network algorithm that we implement needs to store about n $V\times V$ matrices. Therefore, to carry out the proposed contraction scheme, we need to store the result of n matrix-vector operations, in addition to performing the contractions between M^m and B^{m+1}. As each vector-matrix contraction has complexity $O(E)$, the computational cost of the first step is $O(nE)$. Each following step has a complexity of $O(E)$, so that the total complexity is $O(nE)$.

Given that the n tensors are equal except for the last, we only need to store one copy of the connectivity matrix to perform all operations. Although we need to store the result of the vector-matrix contractions, it can be stored with a complexity of $O(V^2)$ or, considering that each non zero element corresponds to an edge, it can also be defined as $O(E)$.

With the reuse of intermediate steps we only need to store n vectors of dimension V, so the total space complexity is $O(nV+E)$.

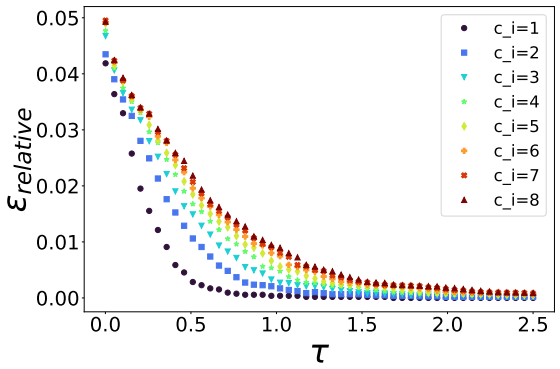

Figure 6: Knapsack problem: Relative error $\varepsilon_{relative} = 1 - V(\vec{x}_{tn})/V(\vec{x}_{gr})$ depending on τ and c_i. Each problem has 1000 classes and a capacity of $0.8\sum_{i=0}^{n-1} c_i w_i$.

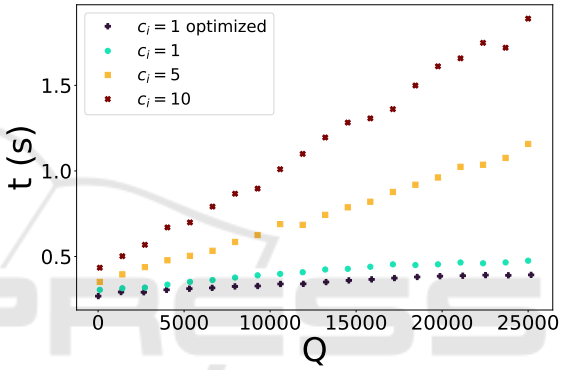

Figure 7: Knapsack problem: Dependency between the capacity of the knapsack Q and the execution time for different problem instances. Use of intermediate calculations, number of classes $n = 5000$, $\tau = 1$.

4 RESULTS AND COMPARATIVES

After explaining the theoretical framework of the algorithms, we present a series of experiments to compare our approaches with other classical algorithms. We evaluate both, the quality of the solutions versus τ and the execution time versus the size of the problem.

4.1 Knapsack Results

The original idea was to compare our algorithm with the Google Or-tools solver. However, due to its limitation of requiring v_i to be integer and its significantly lower maximum weight capacity Q, we have excluded this option. Therefore, the algorithm used for the comparisons is a greedy approach, which is capable of finding a good approximate solution in almost no time for every problem instance. Basically,

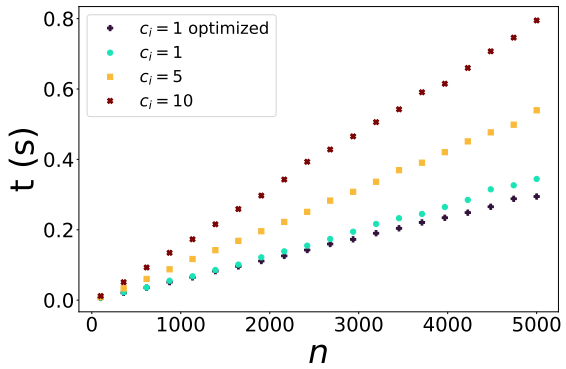

Figure 8: Knapsack problem: Dependency between the number of classes n and the execution time for different problem instances. Use of intermediate calculations, maximum weight capacity $Q = 7000$, $\tau = 1$.

it consists of arranging the items in the knapsack in ascending order according to the value/weight ratio, and then inserting objects until the knapsack is filled to the maximum. Fig. 6 displays how the algorithm performs as a function of τ and the number of elements in all classes (in this case, all classes have the same number of elements). Our indicator is the relative error $\varepsilon_{relative} = 1 - V(\vec{x}_{tn})/V(\vec{x}_{gr})$, being $V(\vec{x}_{tn})$ the value of the tensor network solution and $V(\vec{x}_{gr})$ the value of the greedy solution. As expected, as τ increases, the relative error $\varepsilon_{relative}$ decreases, causing the solution to approximate the greedy approach.

It should be noted that when τ reaches a certain value, the product of the exponential may result in an overflow, causing the algorithm to produce suboptimal solutions. This implies that, although this algorithm is theoretically an exact method, in practice it is not possible to solve arbitrarily large problems. When exponential saturation is reached, the algorithm begins to perform worse and, for this reason, it is crucial to select a value τ that is large enough to distinguish the optimal solution while remaining small enough to prevent saturation.

Additionally, as mentioned in the theoretical section, it is possible to verify in Figs. 7 and 8 that by taking advantage of intermediate calculations, the computational complexity in time of the algorithm depends linearly with the maximum weight capacity Q and the number of classes n. To avoid overflow, it is possible to achieve better results using the Decimal library. In fact, for problems with up to 1000 classes, our algorithm is able to find better solutions than greedy in several problem instances.

4.2 Shortest Path Problem

In order to compare how our algorithm performs against Dijkstra's (Dijkstra, 1959), we propose an experiment where both algorithms have to go from the same origin node to the same destiny node within the same graph. This graph corresponds to Valladolid city (Spain). Given that our algorithm needs to have a fixed number of steps in order to work, we compare how the results evolve according to τ and the number of steps. As can be seen in Fig. 9, both the number of steps and the value of τ influence the result. The higher the number of total steps, the greater τ needs to be in order to reach the optimal solution. Our indicator is the relative error $\varepsilon_{relative} = C(\vec{x}_{tn})/C(\vec{x}_{exact}) - 1$, being $C(\vec{x}_{tn})$ the value given by the tensor network and $C(\vec{x}_{exact})$ the number given by Dijkstra's algorithm. The relative error $\varepsilon_{relative}$ decreases as τ increases, bringing the solution closer to the optimal.

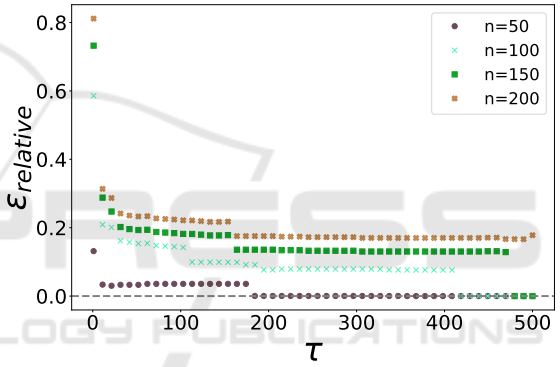

Figure 9: Shortest path problem: Relative error $\varepsilon_{relative} = C(\vec{x}_{tn})/C(\vec{x}_{exact}) - 1$ depending on τ and n. Each problem has $V = 12408$.

5 CONCLUSIONS

This work introduces a quantum-inspired tensor network approach to solving the knapsack and the shortest path problems. The methodology is based on some quantum computing properties, such as superposition, entanglement, or quantum measurements. Theoretically, the proposed algorithms allow to determine the solutions of the problems exactly. However, due to the overflow caused by the implementation, the efficiency of the algorithm is limited by the parameter τ value. Even taking this into account, we have shown that our algorithms are able to find optimal solutions for several problem instances (Figs. 7 and 8).

Future research will focus on analyzing the impact of the parameter τ more systematically and developing alternative implementations that avoid numerical

overflows before reaching the optimal solution. Additionally, we aim to extend this approach to other combinatorial optimization problems, broadening the applicability of quantum-inspired tensor network methods in this domain, and study its possible implementation in quantum hardware to improve its performance.

ACKNOWLEDGMENT

The research has been funded by the Ministry of Science and Innovation and CDTI under ECOSISTEMAS DE INNOVACIÓN project ECO-20241017 (EIFEDE) and ICECyL (Junta de Castilla y León) under project CCTT5/23/BU/0002 (QUANTUMCRIP). This proyect has been funded by the Spanish Ministry of Science, Innovation and Universities under the project PID2023-149511OB-I00.

REFERENCES

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

Ali, A. M., de Leceta, A. M. F., and Rubio, J. L. (2024a). Anomaly detection from a tensor train perspective.

Ali, A. M., Delgado, I. P., and de Leceta, A. M. F. (2024b). Traveling salesman problem from a tensor networks perspective.

Ali, A. M., Delgado, I. P., Markaida, B. G., and de Leceta, A. M. F. (2024c). Task scheduling optimization from a tensor network perspective.

Ali, A. M., Delgado, I. P., Roura, M. R., and de Leceta, A. M. F. (2024d). Polynomial-time solver of tridiagonal qubo and qudo problems with tensor networks.

Bateni, M., Hajiaghayi, M., Seddighin, S., and Stein, C. (2018). Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 1269–1282, New York, NY, USA. Association for Computing Machinery.

Benson, B. M., Ingman, V. M., Agarwal, A., and Keinan, S. (2023). A cqm-based approach to solving a combinatorial problem with applications in drug design.

Biamonte, J. (2020). Lectures on quantum tensor networks.

Biamonte, J. and Bergholm, V. (2017). Tensor networks in a nutshell.

Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Egor, B., Egor, G., and Daria, L. (2022). Two heuristics for one of bin-packing problems. *IFAC-PapersOnLine*, 55(10):2575–2580. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615.

Glover, F., Kochenberger, G., and Du, Y. (2019). A tutorial on formulating and using qubo models.

Golub, G. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, 2(2):205–224.

Gutin, G. and Yeo, A. (2007). The greedy algorithm for the symmetric tsp. *Algorithmic Operations Research*, 2(1).

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

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Katangur, A., Pan, Y., and Fraser, M. (2002). Message routing and scheduling in optical multistage networks using simulated annealing. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 8 pp–.

Korte, B. and Vygen, J. (2000). *Bin-Packing*, pages 407–422. Springer Berlin Heidelberg, Berlin, Heidelberg.

Korte, B. and Vygen, J. (2008). *The Traveling Salesman Problem*, pages 527–562. Springer Berlin Heidelberg, Berlin, Heidelberg.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

Lenstra, J. K. (1992). Job shop scheduling. In Akgül, M., Hamacher, H. W., and Tüfekçi, S., editors, *Combinatorial Optimization*, pages 199–207, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mathews, G. B. (1896). On the partition of numbers. *Proceedings of the London Mathematical Society*, s1-28(1):486–490.

Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O'Brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213.

Pilcher, T. (2023). A self-adaptive genetic algorithm for the flying sidekick travelling salesman problem.

Qing, Y., Li, K., Zhou, P.-F., and Ran, S.-J. (2024). Compressing neural network by tensor network with exponentially fewer variational parameters.

Wang, B. and Dong, H. (2024). Bin packing optimization via deep reinforcement learning.

Wang, J., Roberts, C., Vidal, G., and Leichenauer, S. (2020). Anomaly detection with tensor networks. *CoRR*, abs/2006.02516.