Task Scheduling Optimization with Direct Constraints from a Tensor Network Perspective

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

Iñigo Perez Delgado† and Beatriz García Markaida‡
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: August 4, 2025)

This work presents a novel method for task optimization in industrial plants using quantum-inspired tensor network technology. This method obtains the best possible combination of tasks on a set of machines with directed constraints. With this method, an exact and explicit solution of the problem is provided. This algorithm constructs a tensor network representation of the tensor which provides the solution of the problem. This method is improved in order to reduce the computational complexity of the solution computation, using problem preprocessing, new techniques of condensation of logical constraints, optimization of the value determination technique with previously calculated results, reuse of intermediate computations, and iterative relations for constraints. Three algorithms for computation are presented: the main algorithm, the iterative algorithm which adds only the minimal amount of necessary constraints, and the genetic algorithm which combines the iterative algorithm with basic genetic algorithms. Finally, a simple version of both algorithms was implemented, and their performance was tested, all publicly available.

Keywords: Tensor networks, Combinatorial Optimization, Constrained Optimization, Scheduling Optimization, Quantum-inspired

I. INTRODUCTION

The distribution of production resources is widely considered one of the most interesting and useful problems in industry [1–3]. This distribution problem can be stated as a constraint combinatorial problem, and the time required to obtain the exact optimal solution usually scales exponentially with the size of the problem. However, in real-world applications usually it is not required to obtain the optimal solution, only a good enough solution computed in a reasonable time is needed. There exist some classical heuristic approximations which are capable to reduce the computational time so much, which has led them to be considered great applied methods, such as *genetic algorithms* [4] or *particle swarm optimization* [5]. Even with these methods, obtaining a compatible and optimal solution may be extremely costly for certain types of problem.

Quantum computing applied to industrial cases has become very interesting because of its computational power. Some of the best known and most promising quantum algorithms for combinatorial optimization are the *Quantum Approximate Optimization Algorithm* (QAOA) [6], *Variational Quantum Eigensolver* (VQE) [7], and *Quantum Annealing for Constrained Optimization* (QACO) [8]. However, these algorithms are limited due to the current *Noisy intermediate-scale quantum* (NISQ) state of small quantum computers with notable noise.

Due to this, great expectations have arisen with quantum-inspired methods, which are based on imitating certain quantum processes in classical systems to improve their performance. An example is *digital annealing* to solve *Quadratic Unconstrained Binary Optimization* (QUBO) problems [9]. Another branch inspired by quantum theory is that of *tensor networks* [10], a classical technology based on the use of linear algebra that allows to classically simulate quantum systems both exact and approximately and compress information efficiently. Several algorithms are available in tensor networks to address various combinatorial optimization problems [11–13]. However, it is desirable to have specialized algorithms for particular cases in order to reduce their computational complexity and memory requirements as much as possible and improve their performance.

A specific optimization problem for industrial processes is to assign tasks to be performed to a set of machines, given a set of constraints on the tasks that can be performed by one machine depending on the task performed by a different machine. This case is interesting because both the execution times of the tasks and the constraints between them can be extracted from a historical record of the corresponding manufacturing plant,

* alejandro.mata.ali@gmail.com
† iperezde@ayesa.com
‡ b.garcia@ibermatica.com
§ aitormoreno@lksnext.com

without having to logically deduce them or having to perform tests.

This paper presents an equation that exactly solves the problem and two algorithms to compute it efficiently in an approximate way. It is called the *Directed Constraint Task Scheduler Tensor Network Solver* (DCT-STN). In this approach,the quantum-inspired tensor networks formalism *MeLoCoToN* [14] is combined with iterative methods and genetic algorithms. According to the available information, this is the first work to apply MeLoCoToN to constraint optimization problems and perform a case of *Motion Onion*, described in the same MeLoCoToN work. The main novelty contributions of this work are as follows:

- An exact and explicit tensor network-based equation that solves the problem of minimal makespan with a set of directed constraints.
- An iterative approximated algorithm to compute the solution given by the equation, which in the limit is an exact algorithm.
- A genetic algorithm to compute approximate solutions to the problem.
- Performance analysis with an available Python implementation.

This work is structured as follows. First, Sect. II describes the problem to be addressed and a brief background of the state-of-the-art approach to solving similar problems. Second, Sec. III introduces the algorithm to efficiently obtain tensor network that provides the solution equation for the problem. Third, Sec. IV describes several improvements in the construction of the tensor network to reduce the amount of resources needed for its computation. Then, Sec. V describes how to generalize the algorithm for more general problems. Next, Sec. VI describes the different algorithms to compute the solution from the tensor network construction. Finally, in Sec. VII the performance of the algorithms is tested with several instances and sizes.

All required code is publicly available on the GitHub repository https://github.com/DOKOS-TAYOS/Task_Scheduler_with_Tensor_Networks and a Streamlit application at https://task-scheduler-with-tensor-networks.streamlit.app/.

II. DESCRIPTION OF THE PROBLEM

The problem to solve is the optimal distribution of tasks in a set of machines with some constraints on multiple sets of tasks. That is, the problem has a set of m machines and on i-th machine there are P_i possible tasks, with an execution time T_{ij} for the j-th task on i-th machine, with $i \in [0, m-1]$ and $j \in [0, P_i - 1]$. It also has a set of directed constraints for these task combinations. An example of constraints would be: "If machine 0 performs task 2 and machine 1 performs task 4, machine 2 must perform task 3.". Fig. 1 shows an example of a problem instance, where each machine must do one task, and they are correlated by restrictions.

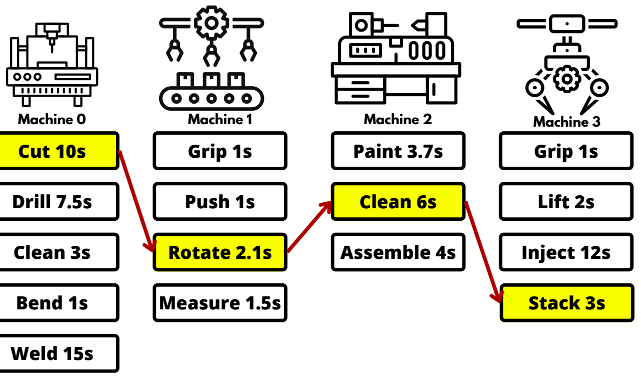


FIG. 1: Example of problem instance with $m = 4$ machines and $P = \{5, 4, 3, 4\}$ tasks, with solution $\vec{x} = (0, 2, 1, 3)$.

The optimal solution is a combination of machines that satisfies the constraints with the lowest execution time. This is essential to increase the productivity of industrial plants. Obviously, other indicators can be used, such as cost instead of execution time. It is assumed that there are no extra execution times due to the order of task execution or moving from one cycle to another. However, the method will be extended in Ssec. V.

The combination of tasks is expressed as a vector $\vec{x}$ such that each element of the vector is the task executed on the machine of the corresponding position. Then, $\vec{x} = (1, 3, 2, 3)$, indicates that machine 0 is assigned task 1, machine 1 is assigned task 3, machine 2 is assigned task 2, and machine 3 is assigned task 3. In other words, x_i is the task executed in the i-th machine.

As with any optimization problem, a cost function is needed. In this case, this function is the total execution time. This can be written as a Tensor quadratic unconstrained discrete optimization [15] problem, with a sum of the individual local costs

$$C(\vec{x}) = \sum_{i=0}^{m-1} T_{i,x_i}. \tag{1}$$

The set of all constraints is obtained by combining all the individual constraints. Each constraint is denoted as R^k, for $k \in [0, n_r - 1]$, which is n_r the number of constraints. The constraints are written as a list with two elements such that the first one is conditional and the second one conditioned. That is, for the above example, "if machine 0 has task 2 and machine 1 has task 4, then machine 2 must have task 3.", the constraint string would be: $R^0 = [[2, 4, ''], [2, 3]]$, where $''$ implies that this machine does not condition the other ones, but it can be the conditioned one.

This problem can be viewed as a version of other scheduling problems such as the Resource-Constrained Project Scheduling Problem (RCPSP) [16, 17], Flow Shop Scheduling Problem [1–3] or Job Shop Scheduling Problem [18]. This type of problem is generally known as NP problems [19–22]. However, because of its importance in industry, there are several methods to solve them in an approximate way. First, genetic algorithms have been widely applied to scheduling problems [1, 4, 23–26], but with a limitation on parameterization of the problems and their size. Another line of solving this kind of problems is the integer programming [18, 27], with limitations in scalability, requiring a large number of variables, and difficulty in the implementation of exotic constraints. The tabu search has also been studied to solve this kind of problem [28, 29] and iterative local search [30, 31], which works in certain contexts, without over-structured instances. Quantum solvers have also been used, such as quantum annealing [32–35] or QAOA [36], with the limitation of current quantum hardware.

III. TENSOR NETWORK EQUATION

The core of the solution equation is inspired by the simulation of a quantum system with qudits by means of a tensor network, taking advantage of the non-unitary operations that the latter allows. However, it is important to recall that this is still a classical formalism, so an explanation from both the classical and quantum computation points of view will be presented. The deduction method relies on improving the algorithm of [13] for the implementation of restrictions to reduce complexity and memory cost, combining it with the tensors presented in [15] for the minimization part, and obtaining the final result with the MeLoCoToN [14] techniques. This means that this equation is a particular case of MeLoCoToN and the first developed for constraint optimization problems.

To check all combinations of tasks, it is necessary to create a tensor ψ, with all elements $\psi_{\vec{x}}$ associated with the possible solution $\vec{x}$. If its elements have a value that decreases with $C(\vec{x})$, and incompatible combinations have zero elements, the problem is reduced to searching for the position of the maximum element of the tensor. It is possible to efficiently construct the tensor network representation of this tensor, and then extract its maximum element position. This tensor network with the extraction method provides the equation that solves the problem.

From a quantum point of view, the construction consists of the following:

1. The creation of the initial uniform superposition state in the qudits: $|\psi^0\rangle = \sum_{\vec{x}} |\vec{x}\rangle$.

2. The application of an imaginary time evolution, so that the amplitude of a combination depends on its cost and the damping constant τ: $|\psi^1\rangle = \sum_{\vec{x}} e^{-\tau C(\vec{x})} |\vec{x}\rangle$.

3. The removal of states by applying constraints by means of projectors: $|\psi^2\rangle = \sum_{\vec{x}} R^0 R^1 \dots R^{n_r-1} e^{-\tau C(\vec{x})} |\vec{x}\rangle$.

4. The measurement and extraction of the basis state with the maximum amplitude of the superposition.

From a classical point of view, it consists of the following:

1. The creation of the initial represented tensor where all elements are equal, representing all possible combinations: $\psi^0_{x_0,x_1,\dots} = 1$.

2. The application of an operation to assign to every possible combination element a value proportional to its cost, with a damping constant τ: $\psi^1_{x_0,x_1,\dots} = e^{-\tau C(\vec{x})}$.

3. The removal of elements of incompatible combinations by means of projective operations: $\psi^2_{x_0,x_1,\dots} = \mathcal{R}^0_{x_0,x_1,\dots} \mathcal{R}^1_{x_0,x_1,\dots} \dots \mathcal{R}^{n_r-1}_{x_0,x_1,\dots} e^{-\tau C(\vec{x})}$.

4. The measurement and extraction of the maximum element position of the represented tensor.

This section is structured as follows. First, Ssec. III A presents the initialization and evolution tensor layers. Second, Ssec. III B presents the creation of the layers of tensors that implement the constraints, and Ssec. III C presents the application of the contraint layers. Then, Ssec. III D presents the value extraction protocol. Finally, Ssec. III E presents the exact and explicit equation from the tensor network.

A. Initial system and imaginary time evolution

The algorithm starts by generating the tensor that considers all possible solutions and assigns a value to them. This is performed in two steps. First, a set of m tensors '+' is created, where the i-th tensor represents the i-th index of the represented tensor ψ^0, associated with the variable x_i. These tensors have all their elements equal to one, and the i-th tensor has P_i elements, each representing each possible value for the variable x_i. The tensor product of all these tensors is equal to

$$\psi^0_{\vec{x}} = +^0_{x_0} +^1_{x_1} \cdots +^{m-1}_{x_{m-1}} = 1. \tag{2}$$

Then, an imaginary time evolution operator is applied to each tensor, obtaining the tensor

$$\psi^1_{\vec{x}} = e^{-\tau C(\vec{x})}. \tag{3}$$

This ensures that the combinations with lower cost have larger elements than the combinations with higher cost. The parameter τ ensures that the values of the elements have a sufficiently large gap for the extraction step, explained in Sec. III D. So, this tensor performs the unconstrained optimization part of the problem. To do this

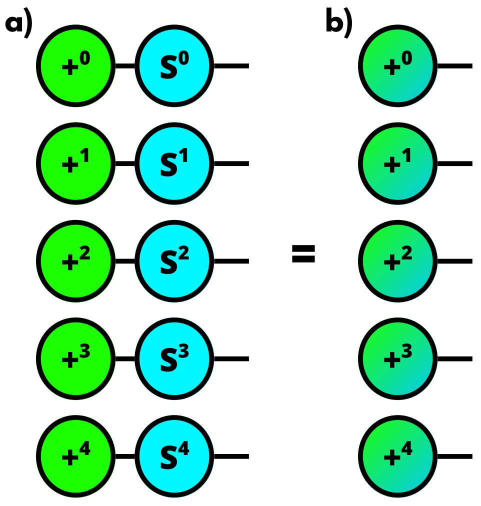


FIG. 2: Initialization nodes. a) Initial superposition layer '+' and evolution layer S. b) New condensated initialization layer '+'.

evolution, the algorithm uses the first two layers of the paper method [15], adapted to this problem cost function by removing the bond indexes in the evolution layer. This layer is shown in Fig. 2 a.

In quantum terms, it makes a superposition of all possible tasks and applies the imaginary time evolution. The state after these steps is

$$|\psi\rangle = \sum_{\vec{x}} e^{-\tau C(\vec{x})}|\vec{x}\rangle, \tag{4}$$

so that the higher the cost of the combination, the lower its amplitude exponentially.

However, since the costs depend only on the variable and not on its neighbors, the 'S' layer of imaginary time evolution can be absorbed in the '+' layer of initialization itself, as shown in Fig. 2 b. The resulting represented tensor is

$$\psi^1_{\vec{x}} = \prod_{i=0}^{m-1} e^{-\tau T_{i,x_i}} = e^{-\tau C(\vec{x})}, \tag{5}$$

being the new '+' tensors

$$+^i_j = e^{-\tau T_{i,j}}. \tag{6}$$

In quantum terms, the minimization operation can also be expressed as

$$\begin{aligned}|\psi^1\rangle =& \sum_{x_0} e^{-\tau T_{0,x_0}}\,|x_0\rangle \otimes \sum_{x_1} e^{-\tau T_{1,x_1}}\,|x_1\rangle \otimes \dots \\ & \dots \otimes \sum_{x_{m-1}} e^{-\tau T_{m-1,x_{m-1}}}\,|x_{m-1}\rangle = \\ =& |+(T)\rangle^0 \otimes |+(T)\rangle^1 \otimes \dots \otimes |+(T)\rangle^{m-1}, \end{aligned} \tag{7}$$

so it can be obtained by means of the tensor product of the $m-1$ vectors already minimized locally;

$$|+(T)\rangle^i = \sum_{x_i} e^{-\tau T_{i,x_i}}\,|x_i\rangle\,. \tag{8}$$

If a task has to be deleted, its corresponding evolution element value has to be replaced with a 0, equivalent to setting its cost to infinity.

B. Constraint layer creation

In order to impose the constraint on the tensor, a set $\{\mathcal{R}^r\}$ of tensor layers is required. Each layer represents a tensor that projects the ψ into the space that satisfies that constraint. That is, the r-th constraint layer sets to zero the values of the elements corresponding to the combinations that do not satisfy the r-th constraint.

The constraint layers can be build as a Matrix Product Operator (MPO), with one tensor connected to each index of the ψ tensor. Each tensor of the MPO checks the value of its corresponding variable and sends it to the next tensor if its value corresponds to the constraint. Finally, the tensor connected with the constrained variable projects the value of the variable into the imposed by the constraint if the other variables have the values that activate it. In other words, if the constraint is $R^0 = [[2,4,\ "],[2,3]]$, the first two tensors of the layer check if the values of its variables are 2 and 4, respectively, and if this is the case, the third tensor multiplies by zero the values associated to values different from 3 in the final variable. In this case, it sets the components $\psi_{2,4,0}, \psi_{2,4,1}, \psi_{2,4,2}, \psi_{2,4,4}$ to zero and $\psi_{2,4,3}$ multiplied by one. This construction requires only sending through the tensors if the first part of the constraint is satisfied, so it can be performed with an MPO of bond dimension equal to 2.

The required tensors have five types, inspired by the Grover oracle circuit [37]: Ctrl, Cctrl, cProj, CcProj, and Id. The Ctrl tensors simply send to the next tensor if the value of its variable is required for the activation of the constraint. The Cctrl tensors perform the same task, but only if they receive the signal that the previous variables have the correct values. The cProj tensors perform the projection only if they receive that the previous variables have the correct values. The CcProj performs the same, but only if they receive the signal by both sides. The Id tensor only passes the signal that it receives.

Following the notation presented in [14] for sparse logical tensors and the index names in Fig. 3, the tensors are defined in the following way.

- $Ctrl^{i,a}$: Ctrl tensor in the i-th variable index, for a constraint with the first part $x_i = a$. Passes 1 by its vertical index if $x_i = a$ and 0 otherwise.

$$\begin{gathered}\mu = j, \quad \nu = \delta_{j,a}, \\ Ctrl^{i,a}_{j\mu\nu} = 1,\end{gathered} \tag{9}$$

  being $\delta_{j,a}$ the Kronecker delta, which equals 1 if $j = a$ and 0 otherwise.

- $Cctrl^{i,a}$: Cctrl tensor in the i-th variable index, for a constraint with the first part $x_i = a$. Passes 1 by

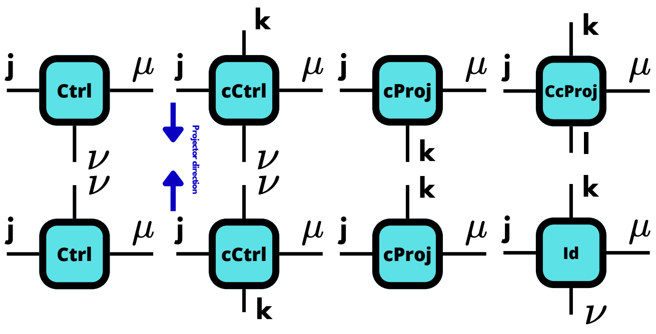


FIG. 3: Name of the indexes for the different tensors depending on the relative position of the projector tensor.

its vertical index if $x_i = a$ and receives 1 from the previous tensor, and 0 otherwise.

$$\begin{aligned} \mu = j, \quad \nu = \delta_{j,a} k, \\ Cctrl^{i,a}_{jk\mu\nu} = 1. \end{aligned} \tag{10}$$

- $cProj^{i,a}$: cProj tensor in the i-th variable index, for a constraint with the second part $x_i = a$ if the first part is satisfied. Only passes a by its horizontal index if it receives 1 from the previous tensor, and all values otherwise.

$$\begin{aligned} \mu = j, \\ cProj^{i,a}_{jk\mu} = (1-k) + k\delta_{j,a}. \end{aligned} \tag{11}$$

- $CcProj^{i,a}$: CcProj tensor in the i-th variable index, for a constraint with the second part $x_i = a$ if the first part is satisfied. Only passes a by its horizontal index if it receives 1 from the previous tensor and the next tensor, and all values otherwise.

$$\begin{aligned} \mu = j, \\ CcProj^{i,a}_{jkl\mu} = (1-kl) + kl\delta_{j,a}. \end{aligned} \tag{12}$$

- Id^i: Id tensor in the i-th variable index. Passes the values of the vertical and horizontal indexes without changes of conditions. It is used for variables that are not included in the constraint.

$$\begin{aligned} \mu = j, \nu = k, \\ Id^i_{jk\mu\nu} = 1. \end{aligned} \tag{13}$$

The constraint layer is constructed by starting the MPO with the Ctrl or cProj tensor at the position of the first variable in the rule, if it is in the first or second part, respectively. Then, include a Cctrl tensor for each variable in the first part of the constraint, except the last one included if it is after the conditioned variable. Include cProj or CcProj in the conditioned variable position if it is after the last variable of the first part or before. Finally, if the last variable of the constraint is in the

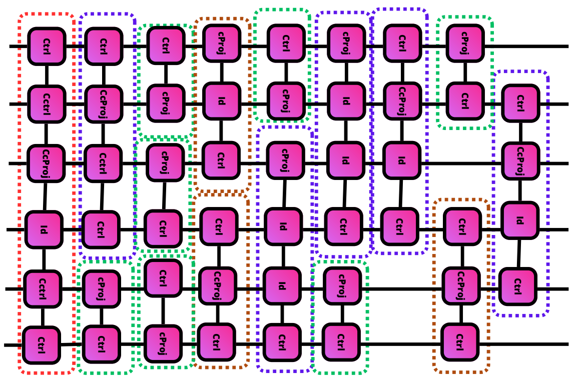


FIG. 4: Example of constraint layers connected in a compact way. The represented constraints are $[[a, a,'','', a, a], [2, a]]$, $[[a,'', a, a,'',''], [1, a]]$, $[['','','','','', a], [4, a]]$, $[[a, a,'','','',''], [1, a]]$, $[['','', a, a,'',''], [2, a]]$, $[['','','','', a, a], [5, a]]$, etc.

first part, include a Ctrl tensor in its position. The other variables with are not in the constraint and are between the variables that are in the constraint have an Id tensor in its positions. Each Ctrl, Cctrl, and cProj have their indexes oriented in a consistent way to give the signals to the projector and the projector receive them. This process is repeated for each constraint, obtaining layers as the ones represented in Fig. 4. Each r-th constraint layer represents the tensor $\mathcal{R}^r$, with elements $\mathcal{R}^r_{\vec{y}^r,\mathcal{S}^r(\vec{x})}$, being $\mathcal{S}^r(\vec{x})$ the subset of variable indices that this constraint layer needs. Due to the construction of these layers, for the element of the represented tensor to be non-zero, it should satisfy that $\vec{y}^r = \mathcal{S}^r(\vec{x})$, and the $\mathcal{S}^r(\vec{x})$ values satisfy the r-th constraint.

C. Constraint operator creation

With the constraints layers set, they are joined one after another, as in Fig. 4, and finally applied to the ψ^1 tensor. It is important that when connecting the constraint layers, the tensor network needs to be as compact as possible to improve its computation. That is, there should be as few intermediate gaps as possible. This will help the contraction algorithms reduce the memory and time costs. The represented global constraint operator is

$$\mathcal{R} = \prod_{r=0}^{n_r-1} \mathcal{R}^r. \tag{14}$$

Applying these tensor layers, the ψ tensor now is

$$\psi^2_{\vec{x}} = \prod_{r=0}^{n_r-1} \mathcal{R}^r_{\mathcal{S}^r(\vec{x}),\mathcal{S}^r(\vec{x})} e^{-\tau C(\vec{x})}. \tag{15}$$

This tensor only has non-zero elements in the compatible combinations, and each element value is exponentially lower if its corresponding cost is larger. So, the position of the larger element of this tensor is th

In quantum terms, the current quantum state is a superposition in which each remaining element satisfies the constraints. This state is

$$|\psi(C,R,\tau)\rangle = \sum_{\vec{x}} \mathcal{R}\; e^{-\tau C(\vec{x})}\, |\vec{x}\rangle\,. \tag{16}$$

D. Variable value extraction

The final step is to extract the position of the largest element of the represented tensor, as this is the one with the lowest cost that satisfies the constraints. It can be done by means of *Half partial traces* [15]. This is based on the fact that the system has a peak value, that is, an element with a value sufficiently larger than the others. Therefore, when a Half partial trace is performed, the maximum at each index should be the same as the global one. This is performed summing over all indices, but the one of the variable to be extracted. If there is a value in the tensor that is large enough, one sum would be larger than the others, and this sum would be the one that includes this element. So, this allows to determine which value of that index contains the desired element, because it is equal to the value of the position of the largest element over this summation.

This is done by connecting a set of nodes with all elements equal to one at the end of the last constraint layer, except for the index corresponding to the variable to be checked, as shown in Fig. 5. This leads to

$$\psi^3_{x_0} = \sum_{x_1,x_2,\dots} \psi^2_{x_0,x_1,x_2,\dots} \tag{17}$$

To satisfy that the tensor has one element so much larger than the other ones, it is necessary that the problem not degenerate, avoiding having two equal peaks, and take the limit $\tau\to\infty$. Then, in the limit

$$\lim_{\tau\to\infty} \frac{\psi^3_{x_0}}{\sum_{x_0}\psi^3_{x_0}} = \lim_{\tau\to\infty} \frac{\sum_{x_1,x_2,\dots} e^{-\tau C(\vec{x})}}{\sum_{x_0,x_1,x_2,\dots} e^{-\tau C(\vec{x})}} = \delta_{x_0,X_0}, \tag{18}$$

being $\vec{X}$ the optimal compatible solution. This means that, in the limit, the tensor has only one non-zero element, and can be extracted by summing over all the other indices.

In the resulting vector, the position of the element whose absolute value is the largest is the correct value of x_0, so the task of the first machine. The same procedure can be performed to determine the other variables, changing only the final layer. To determine the i-th variable, the initialization and constraint layers are connected as before. Now, the vector of ones are connected to each free index, but the one of the i-th variable, the one to be determined.

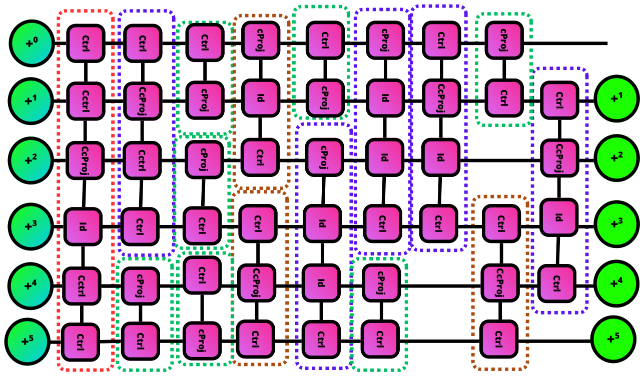


FIG. 5: Tensor network performing the optimization, constraint and traces for determine the first variable.

E. Exact and Explicit Equation

With the construction of the tensor network, each variable can be determined by argmax to the result of the contraction of the tensor network. $\vec{\omega}^i(\tau)$ is defined as the tensor network created to determine the i-th variable, the one that represents ψ^3. Each tensor network is independent of the previous one and depends only on the problem and the parameter τ, so it does not depend on the solution. This leads to the result

$$x_i = \lim_{\tau\to\infty} \arg\max_j \{\omega^i_j(\tau), j\in[0,P_i)\}. \tag{19}$$

Taking into account that a tensor network is an equation, this is also an equation. However, to obtain a more explicit function, the binary expression of each variable can be used. This means that instead of having one variable x_i with P_i possible values, there are $\log_2 P_i$ binary variables $x_{i,k}$. This can be performed by splitting the final free index of $\vec{\omega}^i(\tau)$ into a set of binary indices and performing the Half partial trace with them to determine every binary component of the x_i variable. In this case, because there are only two possible solutions in the limit, $(1,0)$ for 0 or $(0,1)$ for 1, if the resulting vector is multiplied by the tensor $(-1,1)$, if the optimal has the value 0, the result is -1 and $+1$ otherwise. If a variable has P_i that is not a power of two, it is extended with new tasks with infinite cost. This final contraction is defined as $\Omega_i(\tau)$, which is a scalar. Then, if a Heaviside step-function is applied, then the result is exactly the optimal value of that binary part of the variable. This means that

Theorem 1 (Exact and explicit solution equation)**.** *Given a task scheduling problem of m machines defined by a set of times T and a set of directed constraints R, and without degeneration, then its optimal solution is given by the exact and explicit equation*

$$x_{i,k} = \lim_{\tau\to\infty} H(\Omega_{i,k}(\tau)), \tag{20}$$

being $x_{i,k}$ the k-th bit of the task performed in the i-th machine, τ is a decay parameter, $H(\cdot)$ is the Heaviside *step-function and $\Omega_{i,k}(\tau)$ is the resulting tensor network from the connection of the initialization, constraint and trace layers, with the $(-1,1)$ vector connected to the k-th index of the i-th variable.*

IV. IMPROVEMENTS FOR REAL COMPUTATION

The contraction of this tensor network can be made with an algorithm that looks for the most efficient route of contraction or by first contracting all the tensors of the last machine, then this resulting tensor with all those of the previous one, and so on until the whole network is completed. The latter scheme will result, taking into account that all $P_k = P$ for simplicity, in a computational complexity of $O(2^{n_r} n_r P^2 m)$ per variable in the worst case. Even with reuse of intermediate computations as described in [15], the global complexity remains $O(2^{n_r} n_r P^2 m)$. This will depend significantly on the particular case to be solved and the structure of the resulting layers for the constraints, as well as their ordering, so it could have significantly lower complexity.

In order to reduce the number of tensors to use, the number of operations, and the amount of memory to use, several steps of compression of the constraints are applied to the tensor network to be contracted. First, Ssec. IV A presents a pre-processing step to prepare the problem for optimizing the execution of the resolution algorithm. Second, Ssec. IV B presents all the process to condensate the constraints and optimize them for its application for an efficient further contraction. Then, Ssec. IV C describes an improved way to extract the values of the variables taking advantage of the previous variables values.

A. Pre-processing

Before the construction of the tensor network equation, the problem has to be pre-processed. The first step is to normalize the costs. The intention is for all possible costs to be in the range $(-1, 1)$. Thus, the time evolution maintains the represented tensor elements in the range $(e^{-\tau}, e^{\tau})$, avoiding overflows or too small gaps. This helps the computation algorithm not have to adjust τ for each problem manually. This can be done by rescaling all the times of the problem with the sum of the maximum times and the sum of the minimum times for each machine.

The next step is to sort the machines so that the tensor network has the lowest possible number of tensors. This will be better understood in Ssec. IV B. To do this, the machines that appear in more constraints are placed closer together in the most central area of the problem. This means that if machine 0 appears in 3 constraints, machine 1 appears in 2 constraints, machine 2 in 1 constraint, machine 3 in 7 constraints, and machine 4 in 3 constraints, the machines will be sorted in the order $(2, 4, 3, 0, 1)$, changing, for example, the state $\vec{x} = (4, 5, 10, 2, 7)$ to $\vec{x}' = (10, 7, 2, 4, 5)$. The reason for doing this is that machines that appear more in the constraints will have more connections in the tensor network with other machines than those that appear less. In this way, it reduces the number of tensors by eliminating many Id tensors whose only purpose is to connect two distant machines. From now on, all states, times, and constraints are already organized and normalized.

B. Constraints

In this subsection, it is explained how to apply the constraints to the system. First, a rearrangement and condensation of the constraints are performed. The condensation is done to reduce the memory scaling with the number of constraints, and the rearrangement is done to improve the condensation. These two steps are performed to obtain a single layer of bond dimension $d + 1$ that agglomerates d constraints instead of having d layers of one constraint each, which would have a final bond dimension of 2^d. In this way, the bond dimension required can be exponentially reduced in a simple way.

1. Constraint grouping

First, the constraints are grouped into sets that start and end on the same machines and have the same conditioned machine, as this is a condensation requirement. These sets are the constraints that can be condensed in the next step. In case the conditioned machine is before the first conditioning machine, all the constraints that have the same final conditioning machine and that conditioned machine can be joined to the set of the constraints that have the same first conditioning machine. This is the *initial extreme case*. Similarly, in the case the conditioned machine is after the last conditioning machine, all the constraints that have the same initial conditioning machine and that conditioned machine can be joined to the set with the same last conditioning machine. This is the *final extreme case.*

2. Constraint condensation

Within each group, the condensation begins with the generation of subgroups of up to P_k constraints, with k being the first (or last in the final extreme case) conditioning machine in the constraint, so that the conditioning task on the first machine (or last in the final extreme case) is never repeated. This is done to avoid summing in the next step the compatible states differently according to their extremal relation to the constraint. There are actually better constraint compression schemes, using complex techniques, but they are so complicated to generalize that they are left for a possible future study.

3. *Creating constraint layers*

In order to impose such restrictions, a set of constraint layers is created. However, in this case, the signal between them indicates which constraint has been activated. If the first Ctrl of the layer finds that its associated machine is in the task that indicates the third constraint, it will tell the following tensor to verify if its machine is in the task associated with the third constraint. Then, the Ctrl tensor will send the next the signal 3 through its vertical index. If it is not in the value of none of the constraints, the returned signal is zero. If Cctrl receives the signal 3 and its variable is at the corresponding value given by the third constraint, it will return 3 to the following tensor. Otherwise, it will return zero because no constraint is activated. And so on until reaching the cProj, which in case of receiving that all the conditioning machines are in the values of the third constraint, it will force its machine to be in the task of the machine conditioned in this constraint, by means of a projection. The CcProj has controls on both sides, so it has to receive the same signal on both sides to project. If it receives 0 or different values on both sides, the constraints conditions will not have been met, and therefore the projection will not apply.

Tensors are described as:

- $Ctrl^{i,\vec{a}}$: Ctrl tensor in the i-th variable index, for a set of constraints with the first part $x_i = a_r$ for the r-th constraint. Passes $r+1$ by its vertical index if $x_i = a_r$ and 0 otherwise.

$$\begin{gathered} \mu = j, \quad \nu = \sum_r (r+1)\delta_{j,a_r}, \\ Ctrl^{i,\vec{a}}_{j\mu\nu} = 1. \end{gathered} \tag{21}$$

- $Cctrl^{i,\vec{a}}$: Cctrl tensor in the i-th variable index, for a set of constraints with the first part $x_i = a_r$ for the r-th constraint. Passes $r+1$ by its vertical index if $x_i = a_r$ and receives $r+1$ from the previous tensor, and 0 otherwise.

$$\begin{gathered} \mu = j, \quad \nu = \sum_r (r+1)\delta_{j,a_r}\delta_{k,r+1}, \\ Cctrl^{i,\vec{a}}_{jk\mu\nu} = 1. \end{gathered} \tag{22}$$

- $cProj^{i,\vec{a}}$: cProj tensor in the i-th variable index, for a set of constraints with the second part $x_i = a_r$ for the r-th constraint if the first part is satisfied. Only passes a_r by its horizontal index if it receives $r+1$ from the previous tensor, and all values otherwise.

$$\begin{gathered} \mu = j, \\ cProj^{i,\vec{a}}_{jk\mu} = \delta_{k,0} + \sum_r \delta_{k,r+1}\delta_{j,a_r}. \end{gathered} \tag{23}$$

- $CcProj^{i,\vec{a}}$: CcProj tensor in the i-th variable index, for a set of constraints with the second part $x_i = a_r$ for the r-th constraint if the first part is satisfied. Only passes a_r by its horizontal index if it receives $r+1$ from the previous tensor and the next tensor, and all values otherwise.

$$\begin{gathered} \mu = j, \\ CcProj^{i,\vec{a}}_{jkl\mu} = (\delta_{k,0} + \delta_{l,0}) + \sum_r \delta_{k,r+1}\delta_{l,r+1}\delta_{j,a_r}. \end{gathered} \tag{24}$$

- Id^i: Id tensor in the i-th variable index. Passes the values of the vertical and horizontal indexes without changes of conditions. It is used for variables that are not included in the constraint.

$$\begin{gathered} \mu = j, \nu = k, \\ Id^i_{jk\mu\nu} = 1. \end{gathered} \tag{25}$$

This condensation allows the tensor network to be composed of a set of $O(n_r/P)$ constraint layers, with bond dimension $O(P)$. This improves the computational complexity from $O(2^{n_r} n_r P^2 m)$ to $O(P^{n_r/P} n_r m)$. These tensors can be condensed in more complex ways to improve the number of constraints joined, but this could be studied in future research.

C. Extraction reduction

The extraction process can be optimized taking into account that in the i-th variable determination step, the values of the $i-1$ previous variables are known. The part of the tensor network dedicated to the already known variables could be neglected, introducing into the tensors of the current variable the signals they would receive. Then, the same scheme is repeated again, but taking into account only the variables to be determined. To further improve the complexity of each step, the constraint layers that are not needed are removed because they are the bottleneck of the contraction algorithm. Before each determination step, the following modifications are made:

1. All the constraints in which the previous machine conditioned another machine and had to have a different value than the one found are removed. This is because obviously these constraints will never be activated.

2. If the previous machine conditions another machine and its value is the one found, the constraint is kept by eliminating the dependence on the previous machine, since it will always satisfy this part.

3. If the only conditioner of a constraint is the previous machine and has the obtained value, this constraint is eliminated, and the task of the target machine is forced to be the one imposed by the constraint. This is because it will always be active.

This allows to determine the variable without contracting the tensor network.

4. If a constraint has the previous machine as conditional and its value is the obtained, it disappears, since it is always satisfied.

5. If a constraint has as conditioned the previous machine with a value different from the one obtained, the constraint is replaced by one that makes the conditioned combination cannot occur. This is because if that combination is found, the constraint would not be satisfied.

For each variable determined, either by contraction or by this process, the process is repeated for that variable, adding the new known information. That is, in addition to the above, if there is a constraint in which the determined machines are the only conditioners for another machine and they all have the values obtained, this constraint is eliminated, and the task of the other machine is forced to be the one imposed by the constraint. This is because it will always be active.

V. GENERALIZATIONS

This tensor network construction can be adapted to more general problems by performing some modifications. First, if each machine has a cost related to the previous one in the chain, having a nearest-neighbor Tensor Quadratic Unconstrained Discrete Optimization (T-QUDO) cost function

$$C(\vec{x}) = \sum_{i=0}^{m-2} T_{i,i+1,x_i,x_{i+1}}. \tag{26}$$

In this case, the only change is to replace the initialization layer with the initialization and evolution layers of [15].

The second important aspect to be aware is that a certain constraint/constraint set could have additional information. The cost of a particular state can be increased by changing the filtered element in the projector in the corresponding constraint from 1 to exponential with the extra cost and τ. This adds additional terms to the cost in a simple way. Also, the target machine can have a set of tasks instead of a single task by adding more non-null elements to the projectors.

The third generalization is the degenerate case. In this case, the equation can be obtained only with the argmax, making every variable depend on the previously obtained and always choosing the first value obtained in case the argmax returns more than one value. The equation is also exact and explicit, but less elegant. However, in practical situations, it is convenient, because it allows to extract all the M degenerate solutions performing the same algorithm M times, but every time choosing different variable values in the cases where argmax returns more than one value.

VI. COMPUTATION ALGORITHMS

The computation of the solutions from the equation requires to contract a 2D grid network which scales exponentially with the number of constraints. One way to avoid excessive scaling was to condense the constraints, but it is still excessive. To avoid this scaling, two alternative approximate algorithms are proposed: the iterative algorithm in Ssec. VI A and the genetic algorithm in Ssec. VI B. These methods can be combined.

A. Iterative algorithm

This method is based on not applying all the constraints at once at the beginning. Only one compressed constraint is applied, and in case of failure, the first non-satisfied compressed constraint is applied, repeating this process iteratively until all constraints are satisfied. Every constraint removes a piece of the solution space, having its own solution space. When more constraints are applied, the resulting solution space is the intersection of the solution space of all the constraints applied. In this case, the minimum of this new region can be the same as the minimum of the region for a subset of these constraints. Also, if the minimum of a subset is not the same as the minimum of the total set, it will not satisfy all the constraints of the set. This is easy to prove.

Theorem 2 (Minimum for constraint subsets). *Given a set of constraints R and a subset $\hat{R} \subset R$, where X and $\hat{X}$ are the solutions with minimal cost that satisfy their constraints, respectively, $\hat{X}$ satisfies all constraints in R if and only if $\hat{X} = X$.*

Proof.
The sets of combinations that satisfy R and $\hat{R}$ are $\{x \in \mathcal{X}\}$ and $\{\hat{x} \in \hat{\mathcal{X}}\}$. If $\hat{R} \subset R$, then $\mathcal{X} \subset \hat{\mathcal{X}}$. Then, $X \in \hat{\mathcal{X}}$. $\hat{X}$ is the minimum and $\mathcal{X} \subset \hat{\mathcal{X}}$, then $\hat{X} \leq x, \forall x \in \mathcal{X}$. From this point on, all the points can be proven.

- If $\hat{X}$ satisfies R, then $\hat{X} \in \mathcal{X}$. So, $\hat{X}$ is the minimum of $\mathcal{X}$, so $\hat{X} = X$.

- If $\hat{X}$ does not satisfy R, then $\hat{X} \notin \mathcal{X}$. So $\hat{X} = X$ is not possible.

- Trivially, if $\hat{X} = X$, then it satisfies R.

- The claim that if $\hat{X} \neq X$, then $\hat{X}$ cannot satisfy R is proven by contradiction. Let us suppose that $\exists \hat{X} \neq X$ can satisfy R. Then, $\hat{X} \in \mathcal{X}$. As before, this means that $\hat{X}$ is the minimum of $\mathcal{X}$. But the minimum of $\mathcal{X}$ is X. Then, the contradiction appears, proving that if $\hat{X} \neq X$, then $\hat{X}$ cannot satisfy R.

$\square$

Taking the case of all constraints, if the minimum of its associated region is the same as the minimum of the region of some subset of constraints, all the other constraints are irrelevant to the problem, so they can be neglected and obtain the same result. This means that, with fewer constraint layers, the problem can be solved exactly. This is important because the complexity of the contraction algorithm scales exponentially with the number of constraints, so if it can be lower, the algorithm can be performed more efficiently. However, determining the minimal set of constraints that follows this condition is not trivial. For this reason, an approach is to add constraints to the problem until this minimal set is obtained. Verifying that a subset is correct is trivial having its minimum. If the minimum of the subset is the same as the minimum of the total set, it will satisfy all the constraints and will not satisfy some constraints otherwise. Then, adding those constraints not satisfied to the subset allows to, iteratively, complete the minimal subset with the correct minimum.

Then, the optimization algorithm is as follows:

1. The algorithm checks if the simple minimum, the optimal of the unconstrained problem, satisfies all the constraints. If it does, it finishes, meaning that this is the case where the global minimum of the unconstrained problem is the same as the global minimum of the constrained problem by the set of constraints. If not, it continues.

2. The first unmatched constraint is applied, also adding the constraints that are compatible with it (and also unmatched) to condense them, and taking advantage to eliminate more combinations and iterations. If the result satisfies all the constraints, it finishes. If not, this step is repeated adding more unmatched constraints until the result satisfies all constraints or until it reaches the limit of iterations.

3. If it reaches the limit of iterations, that is, the number of sets of condensed constraints allowed to be applied, it finishes with a negative result: no solution that follows all constraints of the chosen set has been found.

It is important to note that this is an algorithm for determining the solution of the minimal set of constraints, not the minimal set itself, due to the extra constraint added to the condensation. From this subset, the minimal subset can be extracted repeating the process without extra constraints condensation. This is the first application of the *Motion Onion* method [14].

In the worst case, this algorithm requires the use of all constraints, which takes much more operations than the simple complete contraction approach. However, if the constraints are redundant and not fine-tuned to be all needed, as in problems as the traveling salesman problem, it would require a small subset of constraints. This is a relevant point for real-world problems, where its constraints are extracted from historical data or heuristics.

B. Genetic algorithm

This algorithm is based on solving several reduced versions of the main problem, called its subproblems, taking advantage that their solutions are computable with more efficiency. Each subproblem considers only a subset of tasks for each machine and a subset of constraints. With this approach, a genetic algorithm can be applied that defines every subproblem as an individual. Each individual in the population has the following attributes:

- Chromosomes: a subset of tasks that each machine can perform and a subset of constraints, consistent with the subset of tasks.

- Result: solution obtained from the tensor network equation and the computation algorithm. This can also be calculated with the iterative method in cases with too many constraints.

- Cost: Cost of the result.

The method is as follows:

1. Random initialization of the population.

2. Computation of the result for each individual. Only a proportion of the best individuals are kept, using the cost function as the evaluator.

3. With these individuals, a crossover is performed. Crosses pairs of parents from the previous generation and corrects the associated constraints, removing the ones that will never be activated.

4. Creation of a set of mutated individuals from the parents and correct the associated constraints. Mutations are performed by changing some tasks to perform in the machines.

5. Check for repeat individuals for their removal.

6. Addition of new randomly created individuals to make up for the missing ones, to have the number of individuals it should have.

7. Addition of new randomly selected constraints for the new generation individuals, until they have a fixed amount. This is performed to make each individual subproblem representative with respect to the main problem.

8. Repetition of steps $2 \rightarrow 6$ until the convergence criteria are met or the fixed maximum number of generations is reached.

The chromosome crossover is performed by exchanging, within the same machine, the possible active tasks a number of times, chosen at random between the two individuals. The constraint correction is applied to each individual each time it is created. If a constraint cannot be activated, it is removed from the individual. For example, if the constraint requires a task that is not included in the individual. For individuals with missing constraints in their heritage, new compatible constraints are added. The output of the algorithm is a set of possible results sorted by their quality.

VII. EXPERIMENTS

The algorithm is tested by creating simulated cases for the three cases of normal algorithm, iterative algorithm, and genetic algorithm with iterative algorithm. In all of them, $\tau = 100$ is enough to obtain the desired results. The algorithms have been implemented in the TensorNetwork library [38], and are performed on the CPU, with an Intel(R) Core(TM) i7-14700HX 2.10 GHz and 16 GB RAM. This implementation is a simplification of the techniques explained in order to analyze the core of the algorithms. It does not perform the iterative removal of constraints and variable tensors during the determination of variable values. It also uses the automatic contraction method provided for the library.

Case generation is performed by choosing the number of machines, the number of tasks per machine, and the number of constraints. First, a list of random times is generated with a uniform distribution between 0 and 10 for each machine and task. Then a set of constraints is created that are compatible with each other, i.e. the same condition cannot lead to two different tasks on the same machine.

First, the runtime of normal and iterative algorithms is tested in Figs. 6 and 7. These tests compare the mean runtime of each algorithm to obtain the solution, with 10 samples per point. Both figures show that the iterative algorithm is much faster than the normal algorithm, even with more calls to the solver. In both cases, Figs. 6c and 7c show that the number of constraints increases the runtime. In Figs. 6a and 6b, normal algorithm does not show a clear behavior for the number of tasks and machines. However, this is because of the limited size of experiments that can be performed due to the high memory scaling of the normal algorithm.

In the iterative case, Figs. 7a and 7b show that the runtime decreases with the number of machines for a fixed number of tasks and constraints. This is not because of algorithm speed. This behavior is due to the fact that with more machines it is more simple that the constraints do not restrict the solution space enough to require all the constraints.

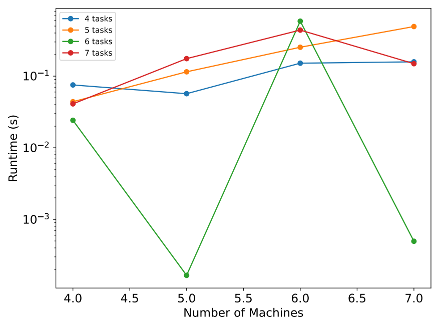
Another interesting experiment is to determine the scaling of the number of required steps in the iterative algorithm to obtain the correct solution. Figs. 8 show the number of steps required for the previous experiments. Figs. 8a and 8b show that the number of steps decreases with the number of machines and tasks, due to the same reasons presented before. However, Fig. 8c shows that the number of steps increases with the number of constraints, faster for a smaller number of machines. This is expected because fewer machines imply more situations where all the constraints are relevant.

Finally, the performance of the genetic algorithm is tested. Fig. 9 shows the success rate of the genetic algorithm to determine a valid combination against the number of machines, for different number of tasks. The number of individuals and generations is set to 30, the maximum number of constraints per individual is set to 10, the number of tasks per individual is set to 4, the proportion of parents is set to 1/3 and the number of crosses and the number of mutations per individual and crossover are set to 5. These parameters are set to have individuals that are more easily executable than the main problem. The larger number of generations and individuals does not appear to improve the performance of the algorithm in the tested instances. The result of the experiments is that the success rate does not seem to depend on the number of tasks, but it increases with the number of machines, probably due to the reasons presented before. However, the success rate is too low, taking into account that these instances are solvable with the iterative method exactly. This indicates that the genetic algorithm does not work properly with this type of problem, at least implemented in the presented way.

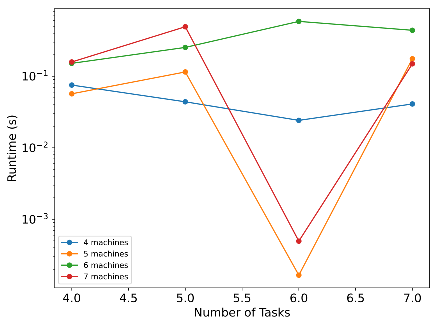
VIII. CONCLUSIONS

This work developed a novel quantum-inspired algorithm for combinatorial optimization and applied it to the industrial case of machine task distribution, combining it with new methodologies. Even with the improvements presented, these algorithms have the limitation of exponential scaling in the worst-case scenario. This situation is not highly probable, but it may be a problem for these kinds of algorithms in production environments. This method has been tested with several instances and problems of various sizes. The iterative algorithm succeeds improving the scalability of the normal algorithm, and the tests shown that in highly random constraint situations, similar to the extracted from historical data, the most of the constraints can be neglected, even more in larger systems. However, the genetic algorithm does not performs well, being unable to obtain compatible solutions. An interesting future line of research is the study of better heuristic and approximated methods for the search for optimal sub-problems to solve the problem. Moreover, its combination with genetic algorithms indicates a possible synergy between both technologies, but it needs to improve the mechanism of combination.

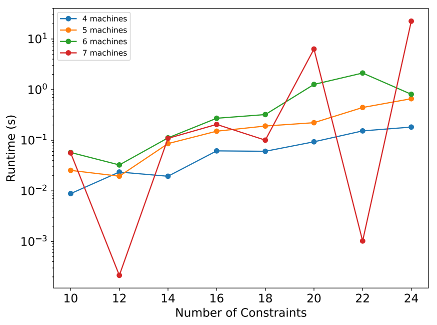
The method can also be extended in future work for other types of bidirectional constraints or restrictions where only certain combinations are possible. This can provide a wider range of applications for this kind of algorithm. Another future study would be to further analyze the algorithm to improve its computation, for example by using Matrix Product State (MPS/TT) compression [39, 40], when constraints are applied or by con-



(a) Runtime against number of machines m, for different number of tasks. There are 15 constraints.

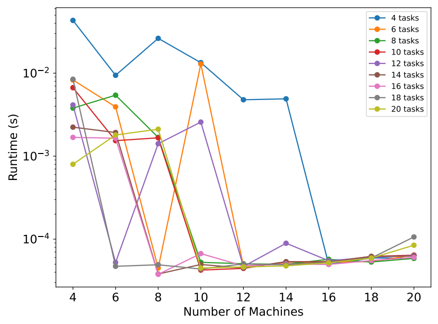


(b) Runtime against number of tasks P, for different number of machines. There are 15 constraints.

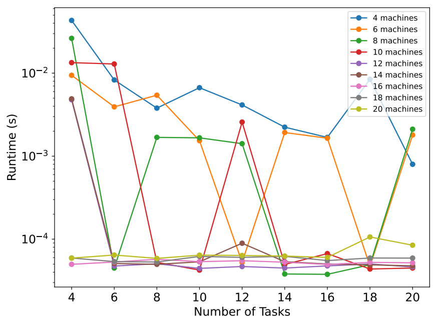


(c) Runtime against number of constraints n_r, for different number of machines. There are 5 tasks.

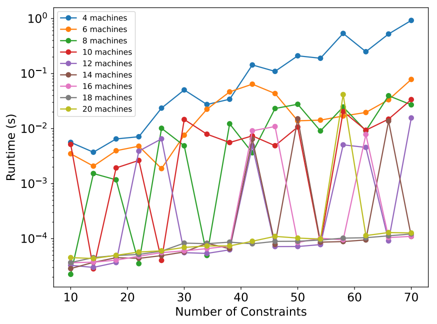
FIG. 6: Scaling of the mean runtime of the normal algorithm with respect to different parameters. There are 10 samples per point and $\tau = 100$.



(a) Runtime against number of machines m, for different number of tasks. There are 25 constraints.



(b) Runtime against number of tasks P, for different number of machines. There are 25 constraints.



(c) Runtime against number of constraints n_r, for different number of machines. There are 5 tasks.

FIG. 7: Scaling of the mean runtime of the iterative algorithm with respect to different parameters. There are 10 samples per point and $\tau = 100$.

densing the constraints in a better way. The general algorithm could also be tested for different significant problems, such as the Traveling Salesman problem or the Job Shop Scheduling problem.

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] Z. Liang, P. Zhong, M. Liu, C. Zhang, and Z. Zhang, A computational efficient optimization of flow shop scheduling problems, Scientific Reports **12**, 845 (2022).

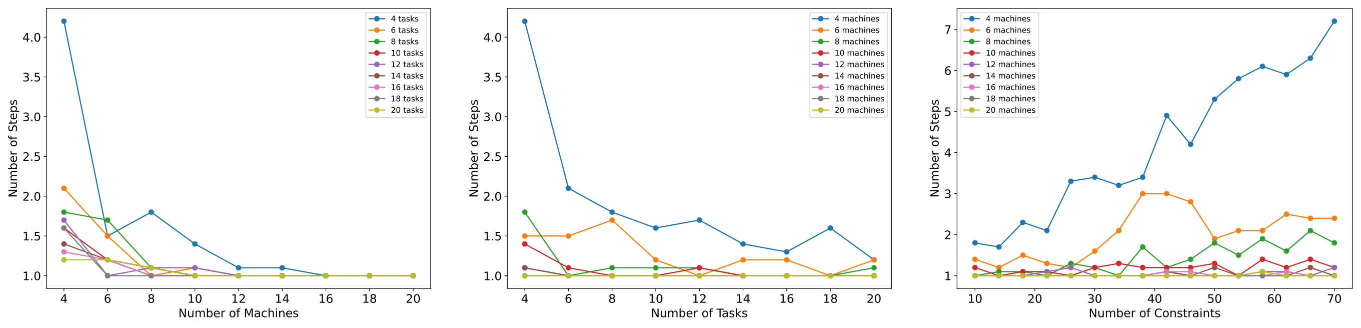
[2] B. A and A. X. M, A simple model to optimize general flow-shop scheduling problems with known break down time and weights of jobs, Procedia Engineering **38**, 191 (2012), iNTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING.

[3] L. Li, S. Liang, Z. Zhu, C. Ding, H. Zha, and B. Wu, Learning to optimize permutation flow shop scheduling via graph-based imitation learning, Proceedings of the AAAI Conference on Artificial Intelligence **38**, 20185 (2024).

[4] M. Pelikan, D. Goldberg, and F. Lobo, A survey of optimization by building and using probabilistic models, in *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36344)*, Vol. 5 (2000) pp. 3289–3293 vol.5.

[5] D. Karaboga and B. Basturk, Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems, in *Foundations of Fuzzy Logic and Soft Computing*, edited by P. Melin, O. Castillo, L. T. Aguilar, J. Kacprzyk, and W. Pedrycz (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007) pp. 789–798.

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



(a) Steps against number of machines m, for different number of tasks. There are 25 constraints.

(b) Steps against number of tasks P, for different number of machines. There are 25 constraints.

(c) Steps against number of constraints n_r, for different number of machines. There are 5 tasks.

FIG. 8: Scaling of the mean number of required steps of the iterative algorithm with respect to different parameters. There are 10 samples per point and $\tau = 100$.

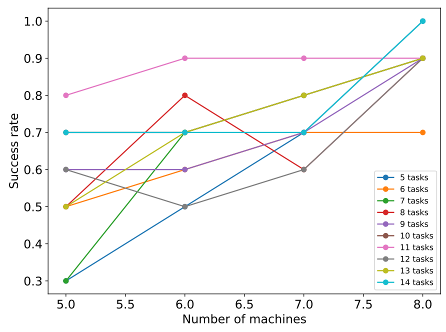


FIG. 9: Mean success rate of the genetic algorithm against the number of machines m for different number of tasks. There are 30 constraints, 10 samples per point and $\tau = 100$.

[7] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, and J. Tennyson, The variational quantum eigensolver: A review of methods and best practices, Physics Reports **986**, 1 (2022), the Variational Quantum Eigensolver: a review of methods and best practices.

[8] I. Hen and F. M. Spedalieri, Quantum annealing for constrained optimization, Phys. Rev. Appl. **5**, 034007 (2016).

[9] 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).

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

[11] K. Sozykin, A. Chertkov, R. Schutski, A.-H. Phan, A. S. CICHOCKI, and I. Oseledets, Ttopt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning, in *Advances in Neural Information Processing Systems*, Vol. 35, edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Curran Associates, Inc., 2022) pp. 26052–26065.

[12] J. Alcazar, M. Ghazi Vakili, C. B. Kalayci, and A. Perdomo-Ortiz, Enhancing combinatorial optimization with classical and quantum generative models, Nature Communications **15**, 2761 (2024).

[13] 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).

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

[15] A. M. Ali, I. P. Delgado, M. R. Roura, and A. M. F. de Leceta, Polynomial-time solver of tridiagonal qubo, qudo and tensor qudo problems with tensor networks (2025), arXiv:2309.10509 [quant-ph].

[16] L. Nedbálek and A. Novák, Bottleneck identification in resource-constrained project scheduling via constraint relaxation, in *Proceedings of the 14th International Conference on Operations Research and Enterprise Systems* (SCITEPRESS - Science and Technology Publications, 2025) p. 340–347.

[17] H. Ding, C. Zhuang, and J. Liu, Extensions of the resource-constrained project scheduling problem, Automation in Construction **153**, 104958 (2023).

[18] W.-Y. Ku and J. C. Beck, Mixed integer programming models for job shop scheduling: A computational analysis, Computers & Operations Research **73**, 165 (2016).

[19] J. Ullman, Np-complete scheduling problems, Journal of Computer and System Sciences **10**, 384 (1975).

[20] R. van Bevern, R. Bredereck, L. Bulteau, C. Komusiewicz, N. Talmon, and G. J. Woeginger, Precedence-constrained scheduling problems parameterized by partial order width, in *Discrete Optimization and Operations Research*, edited by Y. Kochetov, M. Khachay, V. Beresnev, E. Nurminski, and P. Pardalos (Springer International Publishing, Cham, 2016) pp. 105–120.

[21] J. Blazewicz, J. Lenstra, and A. Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Applied Mathematics **5**, 11 (1983).

[22] R. Ganian, T. Hamm, and G. Mescoff, The complexity landscape of resource-constrained scheduling, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20 (2021).

[23] O. Dridi, S. Krichen, and A. Guitouni, Solving resource-constrained project scheduling problem by a genetic local search approach, in *2013 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO)* (2013) pp. 1–5.

[24] F. Pezzella, G. Morganti, and G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & Operations Research **35**, 3202 (2008), part Special Issue: Search-based Software Engineering.

[25] F. Della Croce, R. Tadei, and G. Volta, A genetic algorithm for the job shop problem, Computers & Operations Research **22**, 15 (1995), genetic Algorithms.

[26] T. Murata, H. Ishibuchi, and H. Tanaka, Genetic algorithms for flowshop scheduling problems, Computers & Industrial Engineering **30**, 1061 (1996).

[27] A. Frieze and J. Yadegar, A new integer programming formulation for the permutation flowshop problem, European Journal of Operational Research **40**, 90 (1989).

[28] C. Zhang, P. Li, Z. Guan, and Y. Rao, A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem, Computers & Operations Research **34**, 3229 (2007).

[29] M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, Annals of Operations Research **41**, 231 (1993).

[30] M. Avci, An effective iterated local search algorithm for the distributed no-wait flowshop scheduling problem, Engineering Applications of Artificial Intelligence **120**, 105921 (2023).

[31] A. Ishigaki and S. Takaki, Iterated local search algorithm for flexible job shop scheduling, in *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)* (2017) pp. 947–952.

[32] L. F. Pérez Armas, S. Creemers, and S. Deleplanque, Solving the resource constrained project scheduling problem with quantum annealing, Scientific Reports **14**, 16784 (2024).

[33] P. Schworm, X. Wu, M. Klar, M. Glatt, and J. C. Aurich, Multi-objective quantum annealing approach for solving flexible job shop scheduling in manufacturing, Journal of Manufacturing Systems **72**, 142 (2024).

[34] C. Carugno, M. Ferrari Dacrema, and P. Cremonesi, Evaluating the job shop scheduling problem on a d-wave quantum annealer, Scientific Reports **12**, 6539 (2022).

[35] P. Schworm, X. Wu, M. Glatt, and J. C. Aurich, Solving flexible job shop scheduling problems in manufacturing with quantum annealing, Production Engineering **17**, 105 (2023).

[36] K. Kurowski, T. Pecyna, M. Slysz, R. Różycki, G. Waligóra, and J. Węglarz, Application of quantum approximate optimization algorithm to job shop scheduling problem, European Journal of Operational Research **310**, 518 (2023).

[37] L. K. Grover, A fast quantum mechanical algorithm for database search (1996), arXiv:quant-ph/9605043 [quant-ph].

[38] C. Roberts, A. Milsted, M. Ganahl, A. Zalcman, B. Fontaine, Y. Zou, J. Hidary, G. Vidal, and S. Leichenauer, Tensornetwork: A library for physics and machine learning (2019), arXiv:1905.01330 [physics.comp-ph].

[39] G. Bai, Y. Yang, and G. Chiribella, Quantum compression of tensor network states, New Journal of Physics **22**, 043015 (2020).

[40] M. Lubasch, J. I. Cirac, and M.-C. Bañuls, Unifying projected entangled pair state contractions, New Journal of Physics **16**, 033014 (2014).