Exploration-Exploitation Generative Framework for Constrained Combinatorial Optimization with Tensor Trains (ICML 2026)

Sergei Iudin [1] **Mikhail Podobrii** [1] **Dmitry Zheltkov** [2] **Stanislav Moiseev** [3]

Abstract

Tensor Train (TT) decomposition has emerged as a powerful tool for mitigating the curse of dimensionality in a variety of machine learning and combinatorial optimization tasks. Recent advances show that TT-based generative models can efficiently represent and explore high-dimensional discrete spaces, even when the cost function structure is unknown. However, incorporating hard linear constraints into TT-based optimization remains challenging, as existing interpolation and sampling approaches often fail when feasible points are rare. In this work, we develop a general TT-based framework for constrained combinatorial optimization built upon U(1)-symmetric tensor networks. Our method introduces an expressibility-enhancement mechanism for constructing sparse symmetric tensors and a graph-based diversity measure that guides sample selection during training. Experiments demonstrate that our Julia implementation significantly outperforms existing open-source libraries.

1. Introduction

Tensor Train (TT) decomposition has become a powerful tool in the domains of Artificial Intelligence and Combinatorial Optimization in recent years. The idea of squeezing high-dimensional data via tensor train decomposition originated in the field of quantum mechanics (Affleck et al., 1987; White, 1992), where many-dimensional systems naturally arise from particle interactions. In 2011, Oseledets formalized the approach and brought these ideas into computational mathematics (Oseledets, 2011). To date, TT decomposition was applied to low-rank network layer approximation (Novikov et al., 2015), LLM fine-tuning (Yang et al., 2024), optimization (Antil et al., 2023; Mugel et al., 2022), options pricing (Arenstein & Kastoryano, 2025), and some others (Dolgov & Savostyanov, 2020; Iudin et al., 2025; Sulimov et al., 2017; Sozykin et al., 2026)

This paper focuses on applications of TT in combinatorial optimization. In comparison to classical mixed-integer linear programming (MILP) TT is not limited to linear cost functions and does not use any integrality relaxation. In comparison to Constraint Programming (CP) solvers (Biere et al., 2009) and QAOA (Farhi et al., 2014), the cost function structure may be unknown. Compared with Bayesian optimization (Jones et al., 1998), TT offers better scaling .

We know two paradigms for TT-based optimization. First, we may request cost function values at specific grid points, then interpolate the entire landscape using a low TT-rank assumption (Zheltkov & Osinsky, 2019; Sozykin et al., 2022). Second, we may apply a generative approach (Batsheva et al., 2023): based on some data, learn a probability distribution that encourages low-cost values, and store it as TT. Then we may sample new data from it and use the data for further training until the minimum is found. TT-format speeds up both the training and sampling phases, making high-dimensional distribution tractable.

The challenge of applying TT to combinatorial optimization appears when variables are constrained to a feasible region (e.g., by linear equalities). In such a case, building a TT surrogate for the cost function is difficult due to a large fraction of infeasible grid points. The probability of finding any feasible point is small, and interpolation based on infeasible points is not informative. On the contrary, even a straightforward generative approach may be pre-trained on feasible data, and in a sampling phase, all infeasible samples may be eliminated. However, if the constraints are hard to satisfy, the algorithm's efficiency may decrease significantly. To address the challenge of constraint embedding, the authors in (Nakada et al., 2025) used the method of nilpotent matrices. While effective for some practical cases, it is not generalizable to arbitrary linear equations because of exponential scaling with the number of constraints. In (Ryzhakov & Oseledets, 2023), the authors represent the method of ex-

[1]Moscow Independent Research Institute of Artificial Intelligence (MIRAI), Russian Federation [2]Institute of Numerical Mathematics RAS, Russian Federation [3]T-Technologies. Correspondence to: Sergei Iudin <iudin.s@miriai.org>, Stanislav Moiseev <s.moiseev@t-tech.dev>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

act constraints encoding, but only for constraints that have a particular structure, which is consistent with the tensor train linear topology and locality of interactions. The paper (Lopez-Piqueres et al., 2023) suggests using $U(1)$- symmetric tensor networks for problems with linear constraints. The available feasible dataset was used to build a sparse probability distribution tensor, assigning zero probability to all infeasible vectors. Later, an exact framework for incorporating any linear inequalities into a symmetric TT structure was developed (Lopez-Piqueres & Chen, 2025), with a code available on git (Lopez-Piqueres, 2024a).

Main contributions. We developed a library for constrained optimization, using tensor trains. Our approach is based on a generative $U(1)$-symmetric paradigm, which is the most general and suits any number of linear equality constraints. Two major contributions expand the paradigm: i) We suggested the *expressibility enhancement* step, increasing feasible dataset exploitation. Thus, instead of only two strategies of building a sparse $U(1)$-symmetric tensor from the feasible data, as in the pioneer paper, we suggest a family of such strategies, controlled by a parameter (exploitation power). These strategies outperform the original one. ii) We represented a sparse symmetric tensor train as a graph of $U(1)$ charges, and suggested a graph diversity measure, which helps to choose which samples to use for training on the next algorithm iteration (increasing exploration). Arguably, our library for constrained optimization, implemented in `Julia`, is 100x faster than the primary one, which is available as open source (Lopez-Piqueres, 2024a).

2. Problem Statement

We consider a similar class of optimization problems as in (Lopez-Piqueres et al., 2023):

$$\begin{cases} C(x) \ \rightarrow \ \min, \\ Ax = b, \\ x \in \{0,1\}^N, \end{cases} \tag{1}$$

where C : $\{0,1\}^N \rightarrow \mathbb{R}$ is a black-box objective function, A and b are integer-valued. We emphasize that the domain of the x variable may be, in principle, any finite integer domain; also, A and b may be rational, if they are obtained from an integer by dividing by some scaling factor.

All known to the authors black-box solvers, which assume no structure of the objective function, struggle with hard constraints. E.g., Simulated Annealing (SA) (Kirkpatrick et al., 1983) requires a random step implementation that maintains feasibility. Or constraints may be implemented as penalties of the form $\lambda\,(Ax-b)^2$, but this may be even more problematic, because of the necessity to choose a value for λ and probably a low fraction of feasible solutions evaluated. In this work, we assume that some feasible data for hard constraints is available. This may be due to one of the following reasons: i) There is a feasible dataset sampler available, which may produce diverse feasible samples using some knowledge about the matrix A and vector b. The optimization itself may still be very challenging due to the black-box function $C(x)$, which, in the worst case, takes a long time to evaluate. ii) Feasible space may be explored in some other way, e.g., with a random search / MILP / CP solver, etc.

3. Tensor Train Generative Modeling

Algorithm 1 describes a general generative optimization framework based on tensor-train (TT) probability models. The method maintains a TT-parameterized probability distribution $P(x)$ for integer vector $x = (x_1, \dots x_N)$:

$$P(x) \sim \pi(x),$$
$$\pi(x) = \sum_{\alpha_1,\dots,\alpha_{N-1}} G^{(1)}_{1,\,x_1,\,\alpha_1} G^{(2)}_{\alpha_1,\,x_2,\,\alpha_2} \cdots G^{(N)}_{\alpha_{N-1},\,x_N,\,1}, \tag{2}$$

where relation "$\sim$" means one of the two common situations: $\pi(x)$ is an amplitude, and $P(x) = \frac{|\pi(x)|^2}{Z}$ (where Z is a normalization factor), or $P(x) = \pi(x)$. All TT-cores $G^{(k)}$ are initialized either randomly or as tensors, filled with a constant. The algorithm iteratively refines the probability distribution, using samples drawn from the current model. At each iteration, TT-sampling generates a batch of candidates. A subset of good samples is then selected and merged into the training set, enabling the algorithm to accumulate knowledge about promising regions of the search space. The algorithm assigns importance weights to selected samples, typically using a Boltzmann-type reweighting scheme, and then updates the TT parameters by minimizing a weighted negative log-likelihood loss. After T iterations, the algorithm returns the best found solution.

Some implementation details vary between different realizations of the algorithm, see Section A. For the constrained problem (1), it is much more effective to initialize the tensor with some sparse structure, with zero probabilities for vectors that violate constraints. It helps avoid generating such samples that violate the problem's hard constraints. For this purpose, we exploit properties of $U(1)$-symmetry. Let us define it carefully, but first, give some preliminary definitions. We consider a tensor $\pi_{x_1 \dots x_N}$ for which the following holds:

- Each index x_k has one of two possible orientations (directions): $x_k \in I_{\text{in}}$ or $x_k \in I_{\text{out}}$. We call I_{in} a set of incoming indices and I_{out} a set of outgoing indices.
- Each value of index x_k may be decomposed into the

pair of values $(c_{x_k}, t_{c_{x_k}})$, where we call c_{x_k} a *charge* index, and $t_{c_{x_k}}$ a *degeneracy* index. In this paper, the charges may be integers or integer vectors. More formally, for each k linear space $V^{(k)}$ (associated with index x_k) is a direct sum of charge subspaces:

$$V^{(k)} = \bigoplus_{c_k} V^{(c_k)}. \tag{3}$$

Let $C_{\text{in}} = \sum_{x_i \in I_{\text{in}}} c_{x_i}$, and $C_{\text{out}} = \sum_{x_i \in I_{\text{out}}} c_{x_i}$. We say that the set of index values $(x_1, \dots x_N)$ *conserves the charge* c if $C_{\text{in}} - C_{\text{out}} = c$. Finally, we call the tensor $\pi_{x_1 \dots x_d}$ $U(1)$-symmetric if the following holds for some charge c and all possible values of indices $(x_1, \dots x_N)$:

$$\pi_{x_1 \dots x_N} = (\mathcal{P}_{c_1 \dots c_N})_{t_{c_1} \dots t_{c_N}} \, \delta_{C_{\text{in}}, C_{\text{out}}+c}, \tag{4}$$

where $\delta_{i,j}$ is the Kronecker delta symbol, $c_k = c_{x_k} \quad \forall k$ and $\mathcal{P}_{c_1 \dots c_N}$ is a tensor indexed only by charges. Informally, that means that π becomes block-sparse. If $\pi_{x_1, \dots, x_N}$ is $U(1)$-symmetric, then $\pi_{x_1, \dots, x_N} = 0$ for all sets of values $(x_1, \dots, x_N)$ that do not conserve the charge c. In such cases, c is called a $flux$ of the tensor. We will also call (4) a *charge conservation law*.

Algorithm 1 Generative Optimization with Tensor-Train

Require: Training data $\mathcal{T}$
1: Initialize tensor-train model $P_\theta(x)$
2: **for** $t = 1, 2, \dots, T$ **do**
3: Sample K candidates $S = \{x^{(1)}, \dots, x^{(K)}\} \sim P_\theta$
4: Select $\mathcal{T}_{\text{new}} \subset S$, $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_{\text{new}}$
5: Compute probability weights w_i for $x^{(i)} \in \mathcal{T}$
6: Update θ by minimizing:

$$\mathcal{L}(\theta) = -\sum_i w_i \log P_\theta(x^{(i)})$$

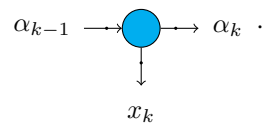
7: **end for**
8: **return** best sample found in $\mathcal{T}$

3.1. Generative modeling with $U(1)$-symmetry

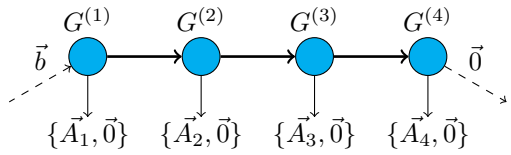
The rigorous description of a generative algorithm based on $U(1)$-symmetric tensor trains may be found in (Lopez-Piqueres et al., 2023). All crucial steps from Algorithm 1 remain the same, but require less memory for storing P_θ, as only nonzero blocks are stored. Also, the big saving of computational resources takes place while calculating the gradients of $\mathcal{L}(\theta)$ and applying gradient descent steps, because all these operations are done on sparse tensors.

In detail, we describe only initialization from the training data. The idea is to associate each variable x_i in problem (1) with a charge $x_i \vec{A}_i$, where $\vec{A}_i$ is the i-th column of the constraints matrix A. Then, the probability amplitude $\pi(x)$ is given as a symmetric TT with the flux b, and each TT-core $G^{(k)}$ is a $U(1)$-symmetric tensor of rank 3. Then, for each TT-core, we choose the direction of the indices. Left (virtual) link index $\alpha_{k-1} \in I_{\text{in}}$, (site) index $x_k \in I_{\text{out}}$, right (virtual) link index $\alpha_k \in I_{\text{out}}$:

$$G^{(k)}(\alpha_{k-1}, x_k, \alpha_k) =$$



By convention, we put the flux on the 1-st TT-core, on the first virtual index of $G^{(1)}$. Thus, the example TT, which encodes the probability distribution for the problem with 4 binary variables, is:



The charge conservation law (4) is then equivalent to constraints in (1), because it says, that $P(x) \neq 0$ only if $\vec{b} - x_1\vec{A}_1 - x_2\vec{A}_2 - x_3\vec{A}_3 - x_4\vec{A}_4 = \vec{0}$.

Building on an exact symmetric TT for the problem (1) may be exponentially hard, because it requires an exponentially large number of link charges (thus, exponentially large rank of tensor train). In (Lopez-Piqueres et al., 2023), heuristic strategies are suggested, which limit the minimum required rank to construct a symmetric tensor train, but also build only the approximation of the required tensor. We describe *Method (i)* in Section B. We chose *Method (ii)* (the most expressive) for this paper as the baseline:

Method (ii): Given the feasible dataset $\mathcal{T}$, for each $x \in \mathcal{T}$, we calculate all the link charges c_{α_k} by propagating: $c_1 = \vec{b}$, $c_{\alpha_i} = c_{\alpha_{i-1}} - x_i\vec{A}_i$. It requires $|b||\mathcal{T}|(N-1)$ operations, where $|\mathcal{T}|$ is the number of (unique) feasible samples, and $|b|$ is the number of equalities in the problem. Then, for each $G^{(k)}$, we search for consistent (with the left link charges $c_{\alpha_{k-1}}$ and right link charges c_{α_k}) site charges $c_{x_k} \in \{\vec{0}, \vec{A}_k\}$. This step requires checking for all $c_{\alpha_{k-1}}$ and c_{x_k}, if $(c_{\alpha_{k-1}} - c_{x_k}) \in \bigcup_{\alpha_k} \{c_{\alpha_k}\}$. We store $\bigcup_{\alpha_k} \{c_{\alpha_k}\}$ as a hash-table based type, thus all the checks take $O(N|b||\mathcal{T}|)$ operations on average, and $O(N|b||\mathcal{T}|^2)$ in the worst case (many hash collisions). TT-rank grows with the dataset size as $O(|\mathcal{T}|)$.

The process of building a nonzero subspace of the tensor

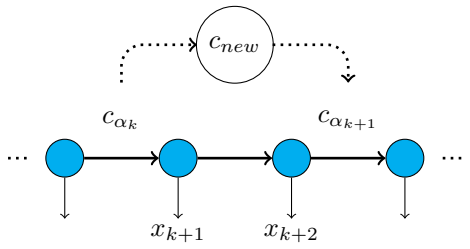


Figure 1. One step of the EELS algorithm for $\gamma = 1$.

train may be interpreted as building a directed acyclic *charge graph*, with nodes corresponding to link charges, edges corresponding to values of x_i, and paths in this graph from the initial node $\vec{b}$ to the final node $\vec{0}$ corresponding to feasible solutions to $Ax = b$. For instance, look at the Figure 2 (a). In these terms, the algorithm of encoding constraints in $U(1)$-symmetric TT is expressive if it generates many feasible paths in the charge graph.

3.2. Expressibility Enhancement Local Search (EELS)

We noticed that all methods for building symmetric TT from data, as suggested in (Lopez-Piqueres et al., 2023), may fail for the same reason. Methods take link charges only from the data but do not apply any strategy to search for unknown charges. If the training data is too small or not diverse enough, then the required link charges, which encode near-optimal solutions, simply do not appear in TT. Moreover, this disadvantage may not be compensated during the training step, because the charge structure pre-defines all samples, which may be sampled from symmetric TT *in principle*, and the learning procedure updates only the probabilities of principally possible samples.

For example, look at the combinatorial problem and dataset $\mathcal{T}$ on Figure 2 (c). Method (ii) fails to find any new samples that do not appear in $\mathcal{T}$. On Figure 2 (a), black circles are the link charges obtained from $\mathcal{T}$, and cyan arrows are all paths in the graph, which encode the dataset samples. Obviously, no new path exists through the black charges, which satisfy the charge conservation law $c_{\alpha_{k+1}} = c_{\alpha_k} - x_{k+1}$ on each edge; thus, no new samples were obtained in TT.

We propose a new parameterized family of methods for initializing symmetric TT, called Expressibility Enhancement Local Search (EELS). First, we initialize symmetric TT with Method (ii). Then, we apply the expressibility enhancement phase. The Figure 1 illustrates the main idea of this phase: we fix the k-th link of TT, and then search for such charges c_{new}, which i) do not appear at the $k+1$-th link; ii) may connect the charges at the k-th link with the charges at the $(k+2)$-th link, satisfying charge conservation law for some x_{k+1}, x_{k+2}. Thus, we "jump" over the $k+1$-th link, and conduct a local search in a charge graph, finding such c_{α_k}, x_{k+1}, x_{k+2}, which satisfy the $U(1)$-symmetry. Sometimes this procedure spawns new charges, which may be assigned to the $k+1$-th link. The length of the "jump" may be regulated with the parameter γ, which is equal to the number of links we jump over. At the Figure 1, $\gamma = 1$, and the Algorithm 2 also provides the detailed description for $\gamma = 1$. Method (ii) may be associated with $\gamma = 0$. For $\gamma > 1$, we search for new charges at the links $k+1$, $k+2$, ..., $k+\gamma$ simultaneously, by considering all possible combinations of c_{α_k}, x_{k+1}, ..., $x_{k+\gamma+1}$, thus, the complexity of the algorithm is $O\left(2^{\gamma+1}|\mathcal{T}||b|(N-\gamma)\right)$ on average. Figure 2 shows that even for $\gamma = 1$, our expressibility enhancement procedure may find some new charges, and thus, new paths in the charge graph, corresponding to feasible samples.

Algorithm 2 EELS algotithm, $\gamma = 1$

Require: $C_1, \ldots, C_{N-1}$ - Sets of link charges, defined from data; A, b - constraints for the problem; $G^{(1)}, \ldots, G^{(N)}$ - TT-cores;

$c_{\alpha_0} \leftarrow \vec{b}$, $c_{\alpha_N} \leftarrow \vec{0}$
for $k = 1$ **to** $N-2$ **do**
  for $c_{\alpha_k} \in C_k$ **do**
    for $x_{k+1} \in \{0, 1\}$ **do**
      $c_{new} \leftarrow c_{\alpha_k} - x_{k+1}\vec{A}_{k+1}$
      for $x_{k+2} \in \{0, 1\}$ **do**
        if $c_{new} - x_{k+2}\vec{A}_{k+2} \in C_{k+2}$ **then**
          $c_{\alpha_{k+2}} = c_{new} - x_{k+2}\vec{A}_{k+2}$
          C_{k+1}.add(c_{new})
          Insert new nonzero blocks:
          $G^{(k+1)} \leftarrow (c_{\alpha_k}, x_{k+1}\vec{A}_{k+1}, c_{new})$
          $G^{(k+2)} \leftarrow (c_{new}, x_{k+2}\vec{A}_{k+2}, c_{\alpha_{k+2}})$
        end if
      end for
    end for
  end for
end for

We emphasize that EELS has nothing in common with regular local search, which may be applied to the data samples to obtain new solution candidates. It acts more like a crossover operator, entangling all the samples from $\mathcal{T}$. On Figure 2(a), EELS obtains the solution, which takes the first bit of the solution 1100, and the last bit from the solution 0011, and the bits 00 in the middle are found using the charge conservation law. Similarly, EELS also finds a feasible solution, taking the first bit from 0011 and the last bit from 1100, filling the middle with a guess of 11. For large datasets, EELS may generate new feasible candidates by mixing all samples from $\mathcal{T}$, producing unpredictable bit concatenations with some "guessed" fillers.

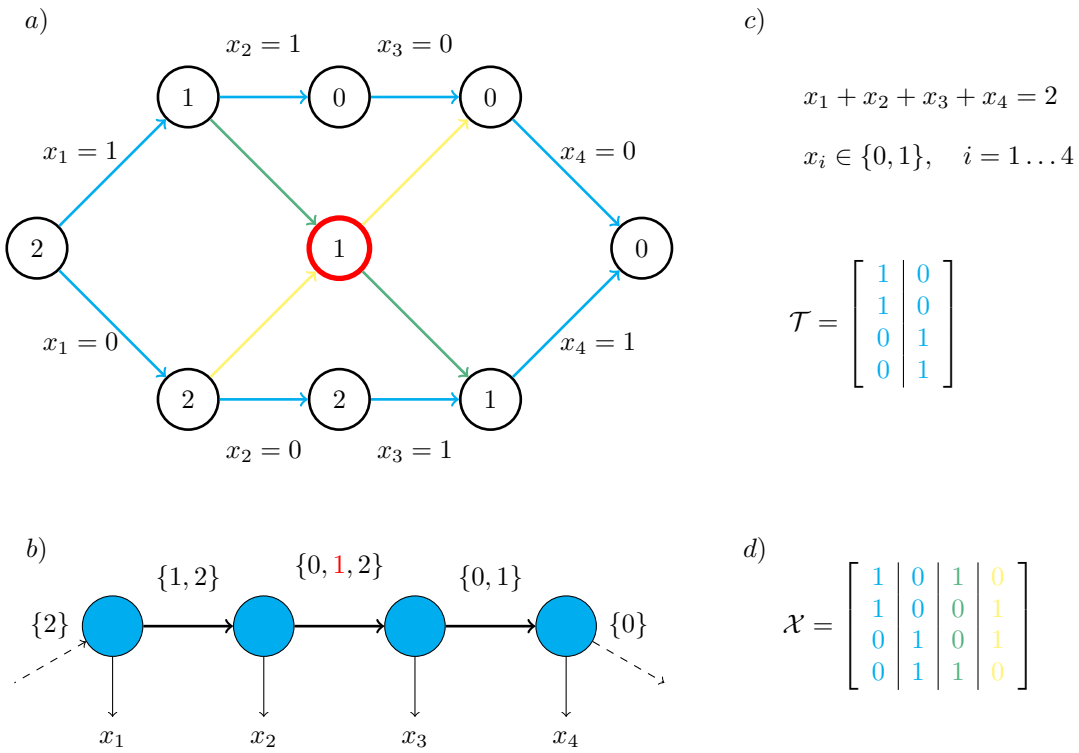


Figure 2. Example problem, where EELS reveals new samples, while standard approaches do not find any new feasible solutions. a) Charge graph of the problem. Cyan paths correspond to the training data, green and yellow lines show new feasible samples after local search. Red charge is a "newborn" charge. b) TT and the link charges nodes, obtained from the training data (black) and found by local search (red). c) Problem constraints and initial feasible dataset. d) All possible unique samples. The sample color in $\mathcal{X}$ matches the color of the corresponding path in the graph.

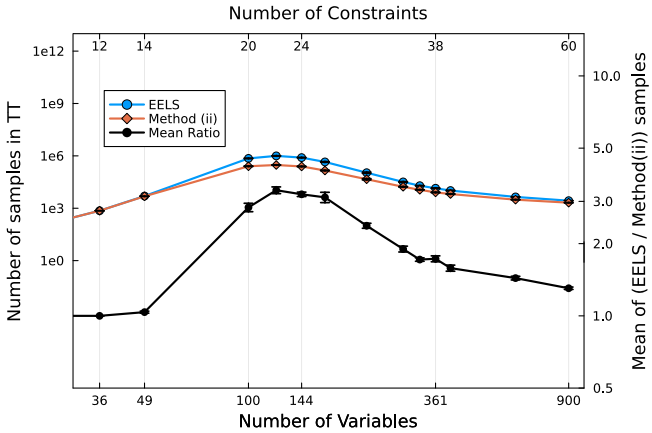
3.3. Amplification of Data Diversity (ADD)

EELS from Section 3.2 is useful when we lack training data or want to better exploit existing data. In this section, we consider the special case of constraints $Ax = b$ when a feasible dataset sampler is available, or it is easy to construct a large dataset $\mathcal{T}$ using extra knowledge. Such situations include, for instance, cardinality constraints $\sum_{i=1}^{N} x_i = \kappa$. In this case, because tensor train optimization trains on the fly, we probably should not use the entire available data but rather pick a good, diverse subset. Otherwise, the complexity of the training step may be (too) large.

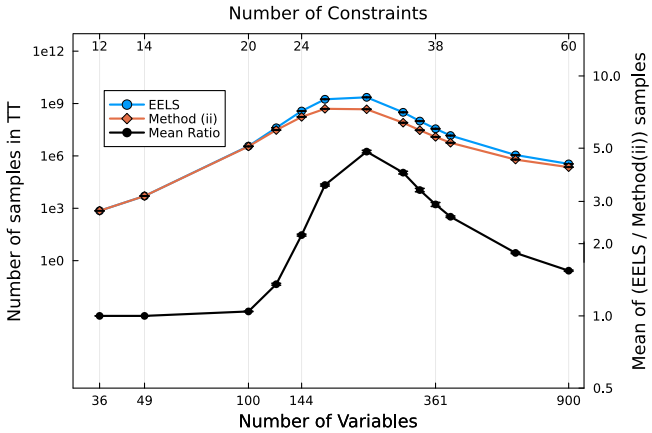
We formulate a problem as follows. Let $\mathcal{T}$ be a data, available at the t-th iteration of the Algorithm 1. This data includes samples from TT and other feasible solutions generated by the feasible data sampler. How to choose a subset $\mathcal{T}' \subset \mathcal{T}$, such that $|\mathcal{T}'| = \mu < |\mathcal{T}|$, to achieve exploration-exploitation trade-off, and keep the complexity of the training step manageable? We need to define the measure of the sample's novelty relative to what the generative scheme has already explored. To do this, we give a formal definition of the charge graph for the problem (1), which we discuss informally in Section 3.2. Let $|b| = m$ be the number of equalities in problem (1) and N the number of binary variables. Let consider a DAG $D = (V, E, \omega, \xi)$, with $\omega : V \to \mathbb{R}^m$ and $\xi : E \to \{\{0\}, \{1\}, \{0, 1\}\}$. Here $\omega(v)$ is a charge of the vertex v, and the set of all vertices consists of several layers: $V = L_0 \sqcup L_1 \sqcup \dots \sqcup L_N$. We call such a graph a charge graph of problem (1) if all of the following stand:

1. $|L_0| = |L_N| = 1, \omega(v_0 \in L_0) = \vec{b}, \omega(v_N \in L_N) = \vec{0}$
2. $(u, v) \in E \Rightarrow u \in L_i,\ v \in L_{i+1}$ for some i
3. If $u_k \in L_k$, and $v_{k+1} \in L_{k+1}$, then $\omega(u_k) - \omega(v_{k+1}) = \vec{0}$ or $\omega(u_k) - \omega(v_{k+1}) = \vec{A_{k+1}}$, where $\vec{A_{k+1}}$ is a k-th column of the matrix A
4. $\xi(e(u_k, v_{k+1})) = \{0, 1\}$, if $\vec{A_{k+1}} = \vec{0}$, else $\xi(e(u_k, v_{k+1})) = \{0\}$ if $\omega(u_k) = \omega(v_{k+1})$, or $\xi(e(u_k, v_{k+1})) = \{1\}$ if $\omega(u_k) - \omega(v_{k+1}) = \vec{A_{k+1}}$.

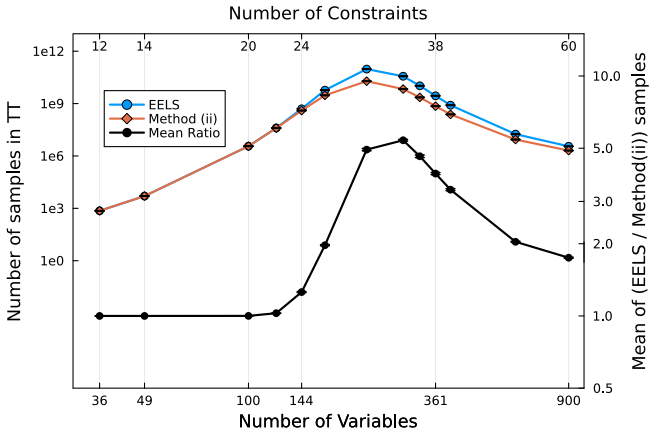
Charge graph on Figure 2(a) satisfies all conditions from the definition. On each path p with edges $(e_1, \dots e_N)$ from



(a) $|T| = 400$.



(b) $|T| = 4000$.



(c) $|T| = 10000$.

Figure 3. Number of unique samples in TT for different $U(1)$-symmetric TT building strategies on the assignment problem.

$v_0 \in L_0$ to $v_N \in L_N$, we define the operation of feasible subset evaluation: $F(p) = \xi(e_1) \times \cdots \times \xi(e_N)$, where "$\times$" is the Cartesian product. We call a charge graph *full* if the set $\mathcal{F}(D) = \bigsqcup_{p \in \mathcal{P}(\mathcal{D})} F(p)$ coincides with all the solutions of $Ax = b$, where $\mathcal{P}(\mathcal{D})$ is a set of all the paths from v_0 to v_N. We define the width of the charge graph as $|D| = \max_{k=0}^{N} (|L_k|)$.

The search for the full graph charge may be exponentially hard, and the same is true for storing such a graph. That is why Method (ii) from Section 3.1 builds only a subgraph $H \subset D$ with a limited width $|H| \leq |\mathcal{T}|$. The goal of the $U(1)$-generative approach is to find such $H \subset D$, that $x^* \in \mathcal{F}(H)$ for the optimal solution x^* of problem (1). The structure of H, which is explored on the training step in Algorithm 1, is completely defined by the selection of samples from the whole available samples set $\mathcal{T}_{\text{new}} \subset S$. The articles (Alcazar et al., 2024) and (Lopez-Piqueres et al., 2023) suggest only the selection of samples with the best cost function $C(x)$, or picking up some rare samples from S for diversity. We suggest using graph distance as a diversity measure. Let $\mathcal{S} = (V^{\mathcal{S}}, E^{\mathcal{S}}, \omega^{\mathcal{S}}, \xi^{\mathcal{S}})$ be a charge graph for some feasible solution $s = (x_1, \ldots, x_N)$. Let $V^{\mathcal{S}} = \bigsqcup_k \{v_k\}$. On each step of the generative optimization, we store a charge graph of already seen samples $\mathcal{H}$, as the sets of charges for each vertex layer: $C_k^{\mathcal{H}} = \bigsqcup_{v \in L_k^{\mathcal{H}}} \{\omega_{\mathcal{H}}(v)\}$. Then the graph distance between $\mathcal{S}$ and $\mathcal{H}$ is:

$$\rho(\mathcal{S}, \mathcal{H}) = \sum_{k=1}^{N-1} \left(1 - \chi(\omega^{\mathcal{S}}(v_k) \in C_k^{\mathcal{H}})\right), \qquad (5)$$

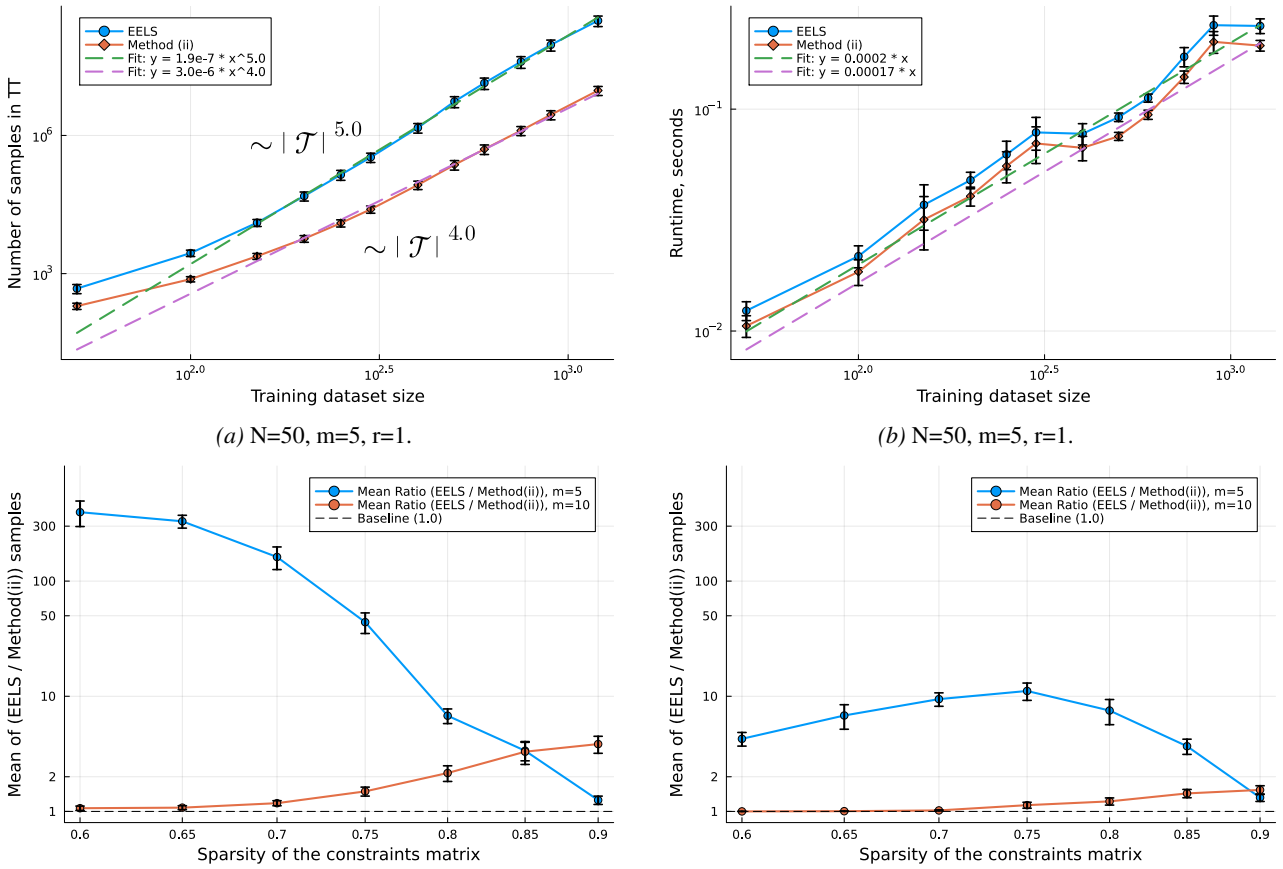
where $\chi(\cdot)$ is an indicator function of the condition in brackets. Thus, we count the number of new charges that appear in the solution s and were not previously seen by the algorithm. In Section 4 we discuss how to balance this novelty measure with exploitation of the near-optimal samples.

4. Results

4.1. Comparison of EELS with other methods

We may compare the expressibility of the methods of building $U(1)$-symmetric tensor trains by fixing the dataset $\mathcal{T}$, defining all link charges degeneracy ranks in (3) $\dim(V^{(c_k)}) = 1$ and then, calculating the number of nonzero elements. If we initialize $G^{(k)}(\alpha_{k-1}, x_k, \alpha_k) = 1$ for all positions, preserving $U(1)$-symmetry, then $\pi(x) = 1$ for all x, conserving charge b. Thus, the number of unique samples in tensor train is just a Frobenius norm squared, which may be calculated efficiently in TT-format.

Figure 4 shows the group of experiments for the problem (1) with the random matrices A of size $m \times n$ and random vector b, both sampled from the uniform distribution on $[-r, r]$. Figure 4a shows the dependency of the number of unique samples in $U(1)$-symmetric TT as the function of



(a) N=50, m=5, r=1.

(b) N=50, m=5, r=1.

(c) N=75, m=5,10; r=1.

(d) N=75, m=5,10; r=2.

Figure 4. Comparison of $U(1)$-symmetric TT building strategies on random problems with N binary variables, m linear equalities and constraints matrix sampled from uniform distribution in range $[-r, r]$. (a) Number of samples dependency on training dataset size. (b) Wall time to build initial $U(1)$-symmetric TT. (c),(d) Num sumpes ratio on matrix sparsity.

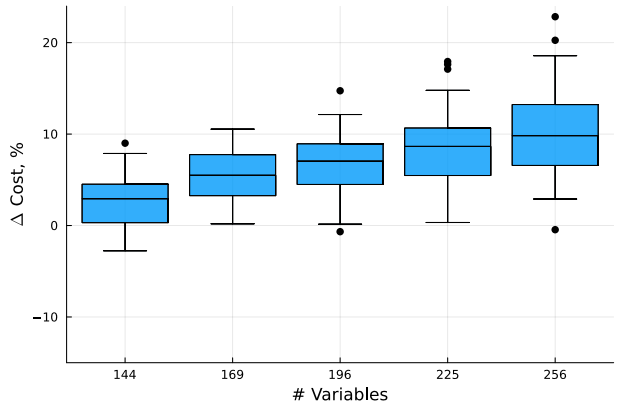
training dataset size. With EELS, the number of unique samples grows as $\sim |\mathcal{T}|^5$ for $r = 1$, while applying only the baseline method (Method (ii) from Section 3.1) encodes only $\sim |\mathcal{T}|^4$ possible feasible solutions. EELS improves the baseline with a small increase in computational time to build an initial $U(1)$-symmetric TT, as shown in Figure 4b. Other experiments on Figure 4 demonstrate that the benefit of using EELS depends a lot on the sparsity of the matrix A, the number of constraints, and the value of parameter r. Generally, if r is big, or the number of constraints is huge, then it is hard to satisfy the if-condition in line 7 of Algorithm 2, and EELS spawns only a small fraction of new link charges c_{new}, if it spawns any at all. In contrast, if the matrix A is sparse, EELS may be an efficient strategy for several dozen constraints. For example, we consider an assignment problem (Koopmans & Beckmann, 1957). This is NP-hard problem for a quadratic/black-box oracle $C(x)$. The constraints matrix of size $2n \times n^2$ is given by the rule:

$$\begin{cases} A_{j,(i-1)n+j} = 1, \quad \forall i, j = 1, \dots, n \\ A_{n+i,(i-1)n+j} = 1, \quad \forall i, j = 1, \dots, n \\ \text{else} \quad A_{k,m} = 0, \end{cases} \tag{6}$$

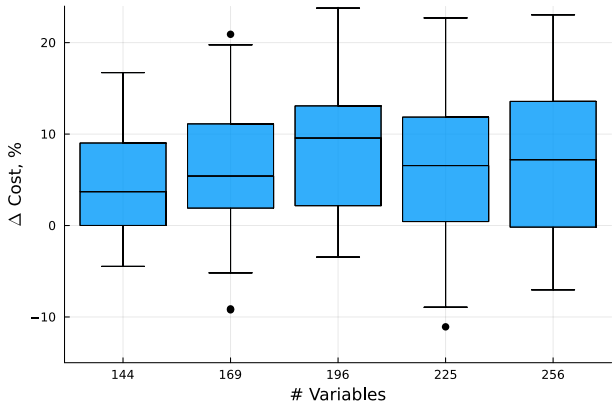
and $b_i = 1 \quad \forall i = 1, \dots, 2n$. The density of matrix A scales as $O(\frac{1}{n})$. Figure 3 shows the group of experiments with the assignment problem. The total number of feasible samples grows monotonically with the problem size as $n!$; however, for large problems, building symmetric TT methods struggle to find new feasible samples when the training data is insufficiently expressive. That is why the number of samples in TT saturates and then decreases. The peak of saturation shifts to the right as the initial dataset size grows. EELS provides approximately $3 - 6$ times more samples at the peak point than Method (ii) from Section 3.1.

4.2. ADD impact on optimization results

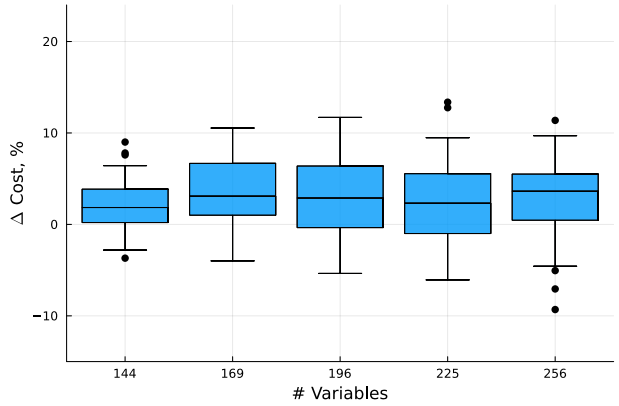
We use the graph distance Equation (5) to balance the novelty of the samples and the exploitation of near-optimal



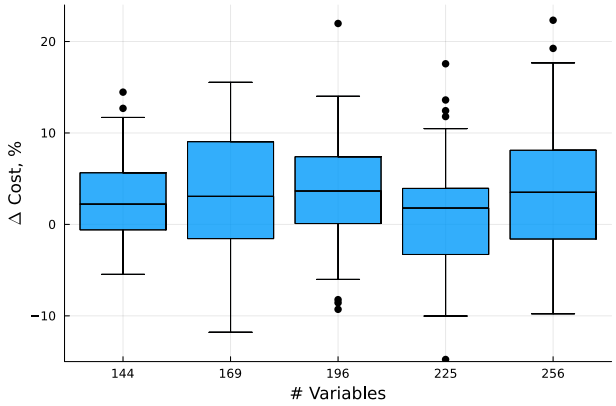
(a) Best Cost + Graph Distance Filter. $w_{\text{filter}} = 0.3$.



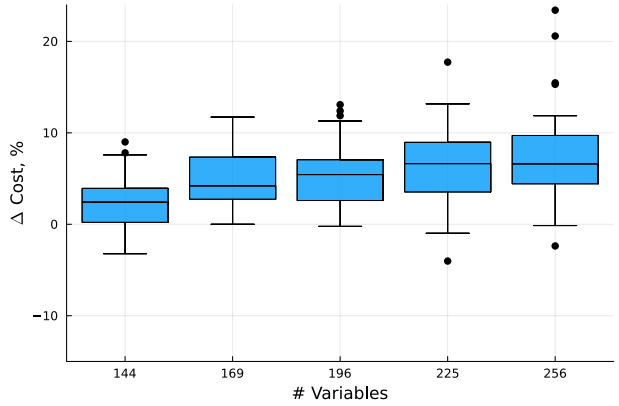
(b) Best Cost + Graph Distance Filter. $w_{\text{filter}} = 0.3$.



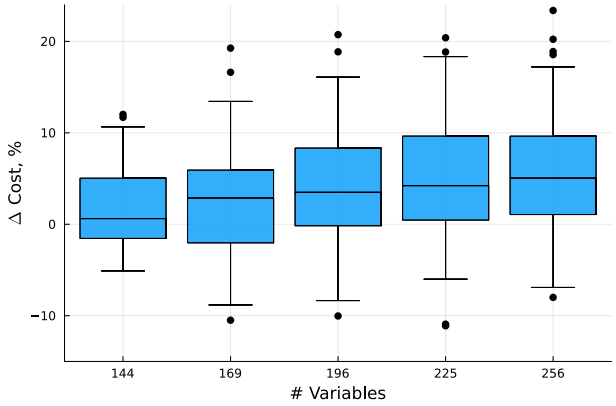
(c) Mixed Selection. $w_{\text{mix}} = 0.4$.



(d) Mixed Selection. $w_{\text{mix}} = 0.4$.



(e) Weighted Rank. $w_{\text{rank}} = 0.7$.



(f) Weighted Rank. $w_{\text{rank}} = 0.7$.

Figure 5. Relative cost function improvement on the assignment problem. Baseline strategy is Best Cost. Each box is obtained from 50 random problems. Parameter tuning is done by grid search. On the left: linear cost function $c^T x$, with c sampled uniformly on $[-50, 50]$. On the right: quadratic cost function $x^T Q x$, with from the uniform distribution on $[-7, 7]$. $|\mathcal{T}| = 1000$, $K = 2 \cdot 10^4$.

samples on the data selection step 4 of Algorithm 1. The assignment problem gives exactly the setting described in Section 3.3: all feasible samples can be generated from permutations of n elements. Thus, it is not a problem to obtain the dataset of feasible samples $\mathcal{T}$, but to choose which data is worth learning. In our experiments, we sample $K/2$ candidates for the dataset S from P_θ, and $K/2$ candidates from the random feasible sampler, and limit the number of samples used for training to μ, because training is an expensive procedure. For some strategies, we also store the graph of all already seen charges $\mathcal{H}$, as a set of charges for each TT link index i. The computation of $\rho(\mathcal{S}, \mathcal{H})$ is then

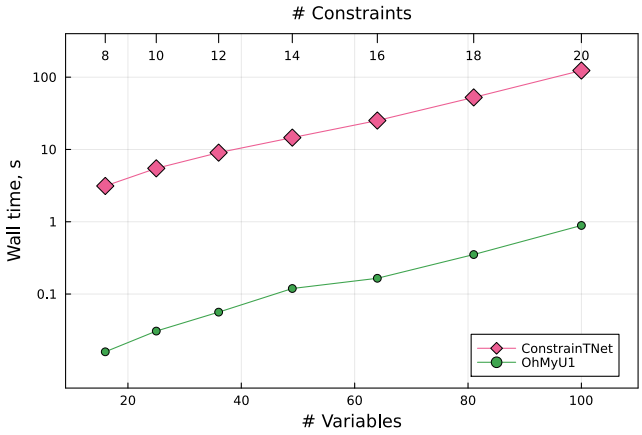


Figure 6. Wall time for assignment problem. Each algorithm performs one iteration, with one sweep, and the training data size is $|T| = 400$. The degeneracy of the block $\dim(V^{(c_k)}) = 1$ for each link index.

requires $O(|b|(N-1))$ operation in average, and storing $\mathcal{H}$ produces extra memory overhead. In our experiments, these parts were not the bottleneck of the whole algorithm; the most intensive step is training, which includes gradient evaluation, sparse core updates, and, for 2-site DMRG, also concatenations and reshapes with $U(1)$-tensors. We suggest three strategies of data selection, and compared them with the baseline from (Lopez-Piqueres et al., 2023):

1. *Best Cost* (Baseline): Pick data from $\mathcal{T}_{\text{new}} \subset S$ with the best cost function $C(s)$ values.

2. *Best Cost + Graph Distance Filter*: Pick $(1-w_{filter})\mu$ samples from $\mathcal{T}_{\text{new}} \subset S$ with the best cost function $C(s)$, then filter remaining data based on condition $\rho(\mathcal{S}, \mathcal{H}) > 0$ and take $w_{filter}\mu$ samples with the best cost. The filtering with $0 < w_{filter} \leq 1$ ensures that any new data that in S will be used for training.

3. *Mixed Selection*: Pick $(1 - w_{mix})\mu$ samples from S based on the best cost, and $w_{mix}\mu$ samples based on the largest values of graph distance $\rho(\mathcal{S}, \mathcal{H})$, $0 < w_{mix} \leq 1$. This strategy picks the "newest" samples, maximizing diversity of the training data.

4. *Weighted Rank*: Pick samples from S based on the lowest weighted rank sum $r_C + w_{rank} r_\rho$, where r_C is a rank of sample based on cost function value ($r_C = 1$ for s with the minimal $C(s)$) and r_ρ is a rank of the sample based on graph distance ($r_\rho = 1$ for the sample s with the maximal $\rho(\mathcal{S}, \mathcal{H})$), $0 < w_{rank} < +\infty$.

Figure 5 demonstrates the relative improvement produced by each of the suggested strategies in comparison to the baseline. We fixed all hyperparameters of the generative scheme, including $|\mathcal{T}|$, K, the feasible sampler, the number of iterations, etc. We used the assignment problem for the benchmarking. All of the suggested strategies are better than baseline, with the median relative improvement achieving up to 10% on the problems with 256 binary variables.

4.3. Benchmarking library performance

The implementation of (Lopez-Piqueres et al., 2023) is not open-source. However, the more recent article by the same authors has a GitHub repo `ConstrainTNet` (Lopez-Piqueres, 2024a). It implements exact approach to building $U(1)$-symmetric TT (no data is needed for initialization, but initialization may be computationally intractable for difficult constraints). Nevertheless, (Lopez-Piqueres, 2024a) also uses `iTensors` (Fishman et al., 2022) dependency. We compared the libraries performance on the assignment problem, excluding the time to build symmetric TT from both methods, since it is done in a fundamentally different way. However, the training and sampling strategies coincide and can be compared directly based on runtime for the same initial tensor. Results are provided on Figure 6. It seems that our library is $\times 100$ faster (we do not use `iTensors` and rely on our own back-end implemented in `Julia`). We tested the script from the example folder of the `ConstrainTNet` git (Lopez-Piqueres, 2024b) (with A and b for the assignment problem) and measured the wall time. The result may be the consequence of our optimizations, aimed at speeding up the computations in a non-degenerate case, with more than a dozen constraints (while `iTensors` aims at more general tensor computations). All our code is available at git (Iudin, 2026).

5. Acknowledgements

We thank Margarita Veshchezerova for providing the initial impetus for this research. We are grateful to Dmitry Gribanov, Elizaveta Iarovikova, Maxim Klimenko, and Ivan Samoylenko for insightful discussions. We appreciate the anonymous reviewers for their helpful feedback.

6. Impact Statement

This work advances methods for black-box constrained combinatorial optimization by leveraging a low-rank symmetric tensor decomposition within the generative Machine Learning approach. The scope of potential applications includes all problems where the cost function structure is unknown and some hard linear constraints must be satisfied. As a result, practitioners in logistics, scheduling, resource allocation, engineering design, and scientific simulation can obtain better solutions faster, lowering computational budgets and accelerating iteration cycles in real-world workflows. We aim to make these benefits accessible: the paper documents algorithms, provides reproducibility artifacts (Iudin, 2026).

References

Affleck, I., Kennedy, T., Lieb, E. H., and Tasaki, H. Rigorous results on valence-bond ground states in antiferromagnets. *Physical review letters*, 59(7):799, 1987.

Alcazar, J., Ghazi Vakili, M., Kalayci, C. B., and Perdomo-Ortiz, A. Enhancing combinatorial optimization with classical and quantum generative models. *Nature Communications*, 15(1):2761, 2024.

Antil, H., Dolgov, S., and Onwunta, A. Ttrisk: Tensor train decomposition algorithm for risk averse optimization. *Numerical Linear Algebra with Applications*, 30(3):e2481, 2023.

Arenstein, L. and Kastoryano, M. Full grid solution for multi-asset options pricing with tensor networks. *arXiv preprint arXiv:2601.00009*, 2025.

Batsheva, A., Chertkov, A., Ryzhakov, G., and Oseledets, I. PROTES: probabilistic optimization with tensor sampling. *Advances in Neural Information Processing Systems*, 36: 808–823, 2023.

Biere, A., van Maaren, H., and Walsh, T. *Handbook of satisfiability*. SAGE Publications Limited, 2009.

Dolgov, S. and Savostyanov, D. Parallel cross interpolation for high-precision calculation of high-dimensional integrals. *Computer Physics Communications*, 246:106869, 2020.

Farhi, E., Goldstone, J., and Gutmann, S. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

Fishman, M., White, S. R., and Stoudenmire, E. M. The ITensor software library for tensor network calculations. *SciPost Phys. Codebases*, pp. 4, 2022. doi: 10.21468/SciPostPhysCodeb.4.

Iudin, S. OhMyU1: Repository for the paper "exploration-exploitation generative framework for constrained combinatorial optimization with tensor-trains", 2026. URL https://github.com/antientropic-guy/OhMyU1_public.

Iudin, S., Veshchezerova, M., Tsarova, K., Tadumadze, G., Shete, V., Hao, J.-K., and Perelshtein, M. Multistart large neighborhood search for the liquefied natural gas transportation and trading over long-term time horizons. *arXiv preprint arXiv:2511.08404*, 2025.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science*, 220(4598): 671–680, 1983.

Koopmans, T. C. and Beckmann, M. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pp. 53–76, 1957.

Lopez-Piqueres, J. Constraintnet.jl: Repository for the paper "cons-training tensor networks", 2024a. URL https://github.com/JaviLoPiq/ConstrainTNet.jl.

Lopez-Piqueres, J. quadratic_knapsack_tn.jl: Example from the constraintnet.jl repository, 2024b. URL https://github.com/JaviLoPiq/ConstrainTNet.jl/blob/master/examples/quadratic_knapsack/quadratic_knapsack_tn.jl.

Lopez-Piqueres, J. and Chen, J. Cons-training tensor networks: Embedding and optimization over discrete linear constraints. *SciPost Phys.*, 18:192, 2025.

Lopez-Piqueres, J., Chen, J., and Perdomo-Ortiz, A. Symmetric tensor networks for generative modeling and constrained combinatorial optimization. *Machine Learning: Science and Technology*, 4(3):035009, 2023.

Mugel, S., Kuchkovsky, C., Sánchez, E., Fernández-Lorenzo, S., Luis-Hita, J., Lizaso, E., and Orús, R. Dynamic portfolio optimization with real datasets using quantum processors and quantum-inspired tensor networks. *Physical Review Research*, 4(1):013006, 2022.

Nakada, H., Tanahashi, K., and Tanaka, S. Quick design of feasible tensor networks for constrained combinatorial optimization. *Quantum*, 9:1799, 2025.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. Tensorizing neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, pp. 442–450, Cambridge, MA, USA, 2015. MIT Press.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Ryzhakov, G. and Oseledets, I. Constructive tt-representation of the tensors given as index interaction functions with applications. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=yLzLfM-Esnu.

Sozykin, K., Chertkov, A., Schutski, R., Phan, A.-H., Cichocki, A. S., and Oseledets, I. TTOpt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning. *Advances in neural information processing systems*, 35:26052–26065, 2022.

Sozykin, K., Rybin, N., Chertkov, A., Phan, A.-H., Oseledets, I., Shapeev, A., Novikov, I., and Ryzhakov, G. Global optimization of atomic clusters via physically-constrained tensor train decomposition. *arXiv preprint arXiv:2601.18592*, 2026.

Sulimov, A. V., Zheltkov, D. A., Oferkin, I. V., Kutov, D. C., Katkova, E. V., Tyrtyshnikov, E. E., and Sulimov, V. B. Evaluation of the novel algorithm of flexible ligand docking with moveable target-protein atoms. *Computational and Structural Biotechnology Journal*, 15:275–285, 2017.

White, S. R. Density matrix formulation for quantum renormalization groups. *Physical review letters*, 69(19):2863, 1992.

Yang, Y., Zhou, J., Wong, N., and Zhang, Z. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3161–3176, 2024.

Zheltkov, D. A. and Osinsky, A. Global optimization algorithms using tensor trains. In *International Conference on Large-Scale Scientific Computing*, pp. 197–202. Springer, 2019.

A. TT Generative Modeling implementation details

In Algorithm 1, if the objective function $C(x)$ is computationally expensive, only a fraction of samples may be evaluated, and the remaining samples are assigned probability weights heuristically. The probability distribution for the samples may be replaced with another physically motivated distribution, such as the Fermi-Dirac distribution. For minimizing $\mathcal{L}(\theta)$, usually the Density Matrix Renormalization Group (DMRG) algorithm is used, which optimizes each TT-core alternatingly (2), or optimizes merged TT-cores (2-site DMRG variation). An optional annealing schedule may gradually decrease the temperature parameter to sharpen the distribution around low-cost regions.

B. Strategies of building $U(1)$-symmetric TT

Method (i): Given the feasible dataset $\mathcal{T}$, for each $x \in \mathcal{T}$, we calculate all the link charges c_{α_k} by propagating: $c_1 = \vec{b}$, $c_{\alpha_i} = c_{\alpha_{i-1}} - x_i \vec{A}_i$. It requires $|b||\mathcal{T}|(N-1)$ operations, where $|\mathcal{T}|$ is the number of (unique) feasible samples, and $|b|$ is the number of equalities in the problem. Then, we initialize each TT-core $G^{(k)}$ with non-zero blocks for subspace with charges $(c_{\alpha_{k-1}}, x_k \vec{A}_k, c_{\alpha_k})$.