LinSATNet: The Positive Linear Satisfiability Neural Networks

Runzhong Wang [1 2] **Yunhao Zhang** [1] **Ziao Guo** [1] **Tianyi Chen** [1] **Xiaokang Yang** [1] **Junchi Yan** [1 2]

Abstract

Encoding constraints into neural networks is attractive. This paper studies how to introduce the popular positive linear satisfiability to neural networks. We propose the first differentiable satisfiability layer based on an extension of the classic Sinkhorn algorithm for jointly encoding multiple sets of marginal distributions. We further theoretically characterize the convergence property of the Sinkhorn algorithm for multiple marginals, and the underlying formulation is also derived. In contrast to the sequential decision e.g. reinforcement learning-based solvers, we showcase our technique in solving constrained (specifically satisfiability) problems by one-shot neural networks, including i) a neural routing solver learned without supervision of optimal solutions; ii) a partial graph matching network handling graphs with unmatchable outliers on both sides; iii) a predictive network for financial portfolios with continuous constraints. To our knowledge, there exists no one-shot neural solver for these scenarios when they are formulated as satisfiability problems. Source code is available at https://github.com/Thinklab-SJTU/LinSATNet.

1. Introduction

It remains open for how to effectively encode the constraints into neural networks for decision-making beyond unconstrained regression and classification. Roughly speaking, we distinguish two categories of such constrained problems: *optimization* and *decision*. *Optimization* problems consider explicit objective functions that are directly related to downstream tasks, whereby their optimization forms are usually more complicated. *Decision* problems do not consider the objective of the downstream task, or the downstream task may not have any explicit objectives. It is possible that *decision* problems also have underlying forms, however, their objectives are usually interpreted as "finding a feasible solution nearest to the input". In particular, the *decision* problem can be divided into two cases: i) only judge if there exists a feasible solution or not; ii) output a feasible solution close to an unconstrained input. This paper focuses on the latter case for *decision* problem, and we term it as *satisfiability* problem if not otherwise specified.

Table 1. Comparison of constraint-encoding for neural networks in finding a solution, with/without an explicit objective function. Note the other works on enforcing a certain kind of *satisfiability* e.g. *permute*, *rank*, *match*, (but not boolean-SAT) can be incorporated by the positive linear constraints as fulfilled by our LinSAT.

Paper	Formulation	Constraint type	Exact gradient?
Amos & Kolter (2017)	*optim.*	linear	Yes
Pogančić et al. (2019)	*optim.*	combinatorial	No
Berthet et al. (2020)	*optim.*	combinatorial	No
Wang et al. (2019a)	*optim.*	combinatorial	Yes
Selsam et al. (2019)	*sat.*	boolean-SAT	Yes
Cruz et al. (2017)	*sat.*	permutation	Yes
Adams & Zemel (2011)	*sat.*	ranking	Yes
Wang et al. (2019b)	*sat.*	matching	Yes
LinSAT (ours)	*sat.*	positive linear	Yes

Notably, machine learning has been well adopted in solving both optimization and decision problems, especially for combinatorial optimization (CO) (Bengio et al., 2021) and SAT problem (Guo et al., 2023; Li et al., 2023). It is relatively easy to introduce learning into problem-solving as a building block under the traditional solving framework (Wang et al., 2021b;a), yet it is more attractive to develop a learning-based framework in a more systematic manner. In this regard, reinforcement learning (RL) (Liu et al., 2023) or alternative sequence-to-sequence models (Vinyals et al., 2015) that solve the problem in an auto-regressive way is of prominence adoption, while they are often less efficient for their sequential decision nature. Thus efforts have also been put into one-shot problem solving, and a popular alternative is designing certain penalties in the loss (Karalias & Loukas, 2020) to respect the constraints. Being more thought-provoking, a more aggressive ambition is to develop end-to-end differentiable neural networks whereby the constraints are seamlessly encoded in their architecture, such that the efficiency of neural networks for one-shot solving can be fulfilled.

[1]Department of Computer Science and Engineering, and MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University [2]Shanghai AI Laboratory. Correspondence to: Junchi Yan <yanjunchi@sjtu.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

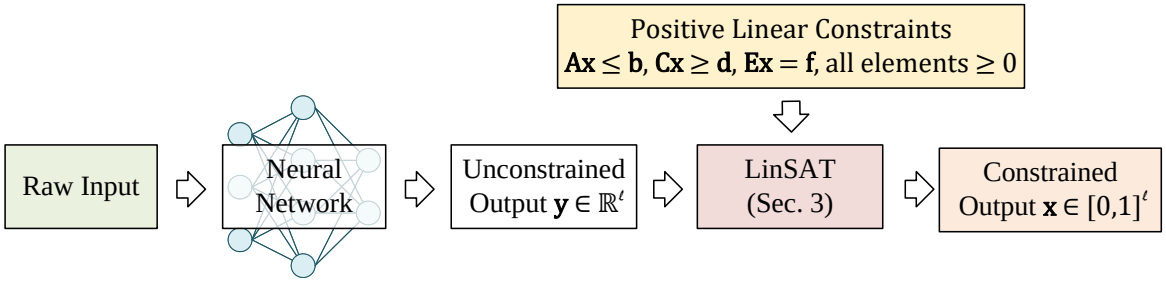


Figure 1. The use case of LinSATNet. If the last layer of the neural network is a linear layer whose output is unconstrained, LinSAT enforces the *satisfiability* of positive linear constraints to the final output.

As shown in Table 1, a number of special constraints have been successfully encoded in neural networks by designing a certain layer for end-to-end training and solving, and as will be shown later, our proposed LinSATNet (i.e. Positive Linear Satisfiability Network) can incorporate the other studied constraints for satisfiability problem and our method is end-to-end differentiable with exact gradient computing. As a side note of Table 1, there exist methods for general combinatorial optimization, e.g. Pogančić et al. (2019); Paulus et al. (2021) over inner-product objectives, while it is often at the cost of being inaccessible to the exact gradient in model training which can potentially hurt the performance.

Our work is technically inspired by the recent success of enforcing the satisfiability of certain constraints to neural networks by deliberately handcrafted layers and in particular the classic Sinkhorn algorithm (Sinkhorn & Knopp, 1967). The Sinkhorn algorithm has been a popular technique in recent constrained neural networks: 1) the permutation constraint to solve jigsaw puzzles by Cruz et al. (2017); 2) the ranking constraint by Adams & Zemel (2011); Cuturi et al. (2019) and the cardinality (top-k) constraint by Xie et al. (2020); Wang et al. (2023); 3) the one-to-one correspondence constraint for graph matching (with outliers at most in one graph) by Wang et al. (2020); Yu et al. (2020); Wang et al. (2022). In this paper, we extend the scope of Sinkhorn algorithm to enforce the satisfiability of the more general positive linear constraints (Luby & Nisan, 1993). Denote $\mathbf{x} \in [0,1]^l$ as the l-dim output, our proposed LinSAT layer jointly enforces the following constraints:

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{C}\mathbf{x} \geq \mathbf{d}, \mathbf{E}\mathbf{x} = \mathbf{f}, \mathbf{x} \in [0,1]^l \tag{1}$$

where all elements in $\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d}, \mathbf{E}, \mathbf{f}$ are non-negative. In fact, this family of constraints incorporates a large scope of real-world scenarios such as the "packing" constraints ($\mathbf{A}\mathbf{x} \leq \mathbf{b}$) and the "covering" constraints ($\mathbf{C}\mathbf{x} \geq \mathbf{d}$), whereby the aforementioned constraints 1)-3) could be viewed as subsets of Eq. (1). The use case of such a positive linear constrained network is illustrated in Figure 1.

In this paper, we first generalize Sinkhorn algorithm to handle multiple sets of marginal distributions. Our multi-set version follows the classic single-set algorithm that is non-parametric (i.e. without trainable parameters for a neural network) and involves only matrix-vector arithmetic operations for exact gradient computing and back-propagation. We theoretically characterize the convergence guarantee and its rate of the proposed algorithm concerning the KL divergence and the L_1 distance to the target marginal distributions. We further show that any positive linear constraints could be equivalently written as multiple sets of marginal distributions, such that the satisfiability of positive linear constraints could be enforced to a differentiable network. **The contributions of the paper are:**

1) We generalize the Sinkhorn algorithm to handle multiple sets of marginal distributions, with the theoretical guarantee that the proposed multi-set algorithm preserves the convergence of the single-set version. Our multi-set algorithm offers theoretical and technical groundings for handling the general positive linear constraints. It may also be of independent interest to the area of matrix normalization.

2) We design LinSAT, a differentiable yet parameter-free light-weighted layer to encode the positive linear constraints, based on our devised multi-set Sinkhorn algorithm. The satisfiability layer only involves matrix-vector arithmetic operations and can be strictly enforced, decoupling the constraint satisfaction from the learning objective. To our best knowledge, this is the first differentiable satisfiability layer addressing the general positive linear constraints.

3) To demonstrate its wide applicability, we deploy our LinSAT to three scenarios regarding constrained routing, outlier-aware matching, and predictive portfolio allocation. In these cases, an explicit objective function is difficult to define and a satisfiable solution is the purpose.

2. Related Work

Sinkhorn Algorithm (Sinkhorn & Knopp, 1967) projects a positive matrix to a doubly-stochastic matrix by alternatively normalizing its rows and columns, and Cuturi (2013) identifies the connection of Sinkhorn and optimal transport. Its effectiveness also motivates recent theoretical studies concerning its convergence rate (Altschuler et al., 2017;

Knight, 2008), whereby Chakrabarty & Khanna (2021) offers a comprehensive theory. Due to its differentiability, Sinkhorn is widely applied in vision (Cruz et al., 2017; Wang et al., 2019b) and learning (Adams & Zemel, 2011; Xie et al., 2020) by enforcing specific constraints. However, as summarized in Table 1, the types of constraints studied in the previous works are less general than the positive linear constraints studied in this paper. Our paper also differs from the existing study of multi-marginal optimal transport (Pass, 2015) since their "multi-marginal" means moving one source distribution to multiple targets, while we are moving multiple sources to multiple targets. To distinguish, we name our algorithm as "Sinkhorn for multi-*set* marginals".

Approximate Solvers for Positive Linear Programming is also an active topic in theoretical computer science. Despite their solid theoretical groundings, this line of works may not be readily integrated into neural networks. For example, Awerbuch & Khandekar (2008); Allen-Zhu & Orecchia (2014) are non-differentiable because their algorithms involve max operations and thresholding functions, respectively. Young (2001); Luby & Nisan (1993) are neither differentiable due to their incremental steps. Another drawback of these methods is that most of them cannot handle a mix of packing ($\mathbf{Ax} \leq \mathbf{b}$) and covering ($\mathbf{Cx} \geq \mathbf{d}$) constraints except for Young (2001). In this paper, we emphasize differentiability to make it compatible with neural networks, and our method could handle any combinations of packing, covering, and equality constraints.

Differentiable Solvers for Constrained Optimization address the problem with objective functions and constraints whereby deep graph matching (Yan et al., 2020; Yu et al., 2020) has been a prominent topic with a quadratic objective. Amos & Kolter (2017) shows the differentiability at optimal solutions via KKT conditions and presents a case study for quadratic programming. Wang et al. (2019a) approximately solves MAXSAT by a differentiable semi-definitive solver. Another line of works develops approximate gradient wrappers for combinatorial solvers: Pogančić et al. (2019) estimates the gradient by the difference of two forward passes; Berthet et al. (2020) estimates the gradient via a batch of random perturbations.

Our approach is devoted to the *satisfiability* setting whereby no explicit objective function is given for the downstream task (Selsam et al., 2019). Note that this is more than just a mathematical assumption: in reality, many problems cannot be defined with an explicit objective function, either due to e.g. the missing of some key variables in noisy or dynamic environments, especially when the objective concerns with a future outcome as will be shown in case studies on partial graph matching (Sec. 5) and predictive portfolio allocation (Sec. 6). However, existing neural networks for *optimization* (e.g. Butler & Kwon (2021) for asset allocation) do not adapt smoothly to these realistic scenarios.

Algorithm 1 Sinkhorn for Single-Set Marginals (Classic)

Input: score matrix $\mathbf{S} \in \mathbb{R}_{\geq 0}^{m \times n}$, single set of marginal distributions $\mathbf{v} \in \mathbb{R}_{\geq 0}^{m}, \mathbf{u} \in \mathbb{R}_{\geq 0}^{n}$.
Initialize $\Gamma_{i,j} = \frac{s_{i,j}}{\sum_{i=1}^{m} s_{i,j}}$;
repeat
  $\Gamma'_{i,j} = \frac{\Gamma_{i,j} v_i}{\sum_{j=1}^{n} \Gamma_{i,j} u_j}$; ▷ normalize w.r.t. $\mathbf{v}$
  $\Gamma_{i,j} = \frac{\Gamma'_{i,j} u_j}{\sum_{i=1}^{m} \Gamma'_{i,j} u_j}$; ▷ normalize w.r.t. $\mathbf{u}$
until convergence

Finally, note that the boolean satisfiability problem (Cook, 1971) also receives attention from machine learning community (Guo et al., 2023), whereby end-to-end neural nets have also been actively developed e.g. NeuroSAT (Selsam et al., 2019) and QuerySAT (Ozolins et al., 2021). As we mentioned in Table 1, the boolean-SAT cannot be covered by our constraint and is orthogonal to this work.

3. Methodology

Sec. 3.1 formulates the classic Sinkhorn algorithm handling a single set of marginal distributions. Sec. 3.2 proposes the generalized multi-set Sinkhorn with a convergence study. In Sec. 3.3 we devise LinSAT layer to enforce the positive linear constraints, by connecting to the marginal distributions.

3.1. Preliminaries: The Classic Sinkhorn Algorithm for Single Set of Marginal Distributions

We first revisit the classic Sinkhorn algorithm in Algorithm 1, which is a differentiable method developed by Sinkhorn & Knopp (1967) to enforce a single set of marginal distributions to a matrix. Given non-negative score matrix $\mathbf{S} \in \mathbb{R}_{\geq 0}^{m \times n}$ and a set of marginal distributions on rows $\mathbf{v} \in \mathbb{R}_{\geq 0}^{m}$ and columns $\mathbf{u} \in \mathbb{R}_{\geq 0}^{n}$ (where $\sum_{i=1}^{m} v_i = \sum_{j=1}^{n} u_j = h$), the Sinkhorn algorithm outputs a normalized matrix $\mathbf{\Gamma} \in [0,1]^{m \times n}$ so that $\sum_{i=1}^{m} \Gamma_{i,j} u_j = u_j, \sum_{j=1}^{n} \Gamma_{i,j} u_j = v_i$. Conceptually, $\Gamma_{i,j}$ means the proportion of u_j moved to v_i. Note that $\Gamma_{i,j}$ usually has no same meaning in the "reversed move" from v_i to u_j if $v_i \neq u_j$[1]. We initialize $\mathbf{\Gamma}^{(0)}$ by

$$\Gamma_{i,j}^{(0)} = \frac{s_{i,j}}{\sum_{i=1}^{m} s_{i,j}}. \tag{2}$$

At iteration t, $\mathbf{\Gamma}'^{(t)}$ is obtained by normalizing w.r.t. the row-distributions $\mathbf{v}$, and $\mathbf{\Gamma}^{(t+1)}$ is obtained by normalizing w.r.t the column-distributions $\mathbf{u}$. $\mathbf{\Gamma}^{(t)}, \mathbf{\Gamma}'^{(t)} \in [0,1]^{m \times n}$ are

[1] This formulation is modified from the conventional formulation where $\Gamma_{i,j} u_j$ is equivalent to the elements in the "transport" matrix in Cuturi (2013). We prefer this formulation as it seamlessly generalizes to multi-set marginals. See Appendix A for details.

scaled by $\mathbf{u}$ before and after normalization. Specifically,

$$\Gamma'^{(t)}_{i,j} = \frac{\Gamma^{(t)}_{i,j} v_i}{\sum_{j=1}^{n} \Gamma^{(t)}_{i,j} u_j}, \quad \Gamma^{(t+1)}_{i,j} = \frac{\Gamma'^{(t)}_{i,j} u_j}{\sum_{i=1}^{m} \Gamma'^{(t)}_{i,j} u_j}. \tag{3}$$

The above algorithm is easy to implement and GPU-friendly. Besides, it only involves matrix-vector arithmetic operations, meaning that it is naturally differentiable and the backward pass is easy to implement by the autograd feature of modern deep learning frameworks (Paszke et al., 2017).

Some recent theoretical studies (Altschuler et al., 2017; Chakrabarty & Khanna, 2021) further characterize the rate of convergence of Sinkhorn algorithm. Define the L_1 error as the violation of the marginal distributions,

$$L_1(\mathbf{\Gamma}^{(t)}) = \|\mathbf{v}^{(t)} - \mathbf{v}\|_1, \; v_i^{(t)} = \sum_{j=1}^{n} \Gamma^{(t)}_{i,j} u_j, \tag{4a}$$

$$L_1(\mathbf{\Gamma}'^{(t)}) = \|\mathbf{u}^{(t)} - \mathbf{u}\|_1, \; u_j^{(t)} = \sum_{i=1}^{m} \Gamma'^{(t)}_{i,j} u_j. \tag{4b}$$

Theorem 3.1 (See Chakrabarty & Khanna (2021) for the proof)**.** *For any $\epsilon > 0$, the Sinkhorn algorithm for single-set marginals returns a matrix $\mathbf{\Gamma}^{(t)}$ or $\mathbf{\Gamma}'^{(t)}$ with L_1 error $\leq \epsilon$ in time $t = \mathcal{O}\left(\frac{h^2 \log(\Delta/\alpha)}{\epsilon^2}\right)$, where $\alpha = \min_{i,j:s_{i,j}>0} s_{i,j}/\max_{i,j} s_{i,j}$, $\Delta = \max_j |\{i : s_{i,j} > 0\}|$ is the max number of non-zeros in any column of $\mathbf{S}$, and recall that $\sum_{i=1}^{m} v_i = \sum_{j=1}^{n} u_j = h$.*

3.2. Generalizing Sinkhorn Algorithm for Multiple Sets of Marginal Distributions

Existing literature about the Sinkhorn algorithm mainly focuses on a single set of marginal distributions. In the following, we present our approach that extends the Sinkhorn algorithm into multiple sets of marginal distributions.

Following Cuturi (2013), we view the Sinkhorn algorithm as "moving masses" between marginal distributions: $\Gamma_{i,j} \in [0, 1]$ means the proportion of u_i moved to v_j. Interestingly, it yields the same formulation if we simply replace $\mathbf{u}, \mathbf{v}$ by another set of marginal distributions, suggesting the potential of extending the Sinkhorn algorithm to multiple sets of marginal distributions. To this end, we devise Algorithm 2, an extended version of the Sinkhorn algorithm, whereby k sets of marginal distributions are jointly enforced to fit more complicated real-world scenarios. The sets of marginal distributions are $\mathbf{u}_\eta \in \mathbb{R}^n_{\geq 0}, \mathbf{v}_\eta \in \mathbb{R}^m_{\geq 0}$, and we have:

$$\forall \eta \in \{1, \cdots, k\} : \sum_{i=1}^{m} v_{\eta,i} = \sum_{j=1}^{n} u_{\eta,j} = h_\eta. \tag{5}$$

Algorithm 2 Sinkhorn for Multi-Set Marginals (Proposed)

Input: score matrix $\mathbf{S} \in \mathbb{R}^{m\times n}_{\geq 0}$, k sets of marginal distributions $\mathbf{V} \in \mathbb{R}^{k\times m}_{\geq 0}, \mathbf{U} \in \mathbb{R}^{k\times n}_{\geq 0}$.
Initialize $\Gamma_{i,j} = \frac{s_{i,j}}{\sum_{i=1}^{m} s_{i,j}}$;
repeat
  for $\eta = 1$ **to** k **do**
    $\Gamma'_{i,j} = \frac{\Gamma_{i,j} v_{\eta,i}}{\sum_{j=1}^{n} \Gamma_{i,j} u_{\eta,j}}$; ▷ normalize w.r.t. $\mathbf{v}_\eta$
    $\Gamma_{i,j} = \frac{\Gamma'_{i,j} u_{\eta,j}}{\sum_{i=1}^{m} \Gamma'_{i,j} u_{\eta,j}}$; ▷ normalize w.r.t. $\mathbf{u}_\eta$
  end for
until convergence

It assumes the existence of a normalized $\mathbf{Z} \in [0, 1]^{m\times n}$ s.t.

$$\forall \eta \in \{1, \cdots, k\} : \sum_{i=1}^{m} z_{i,j} u_{\eta,j} = u_{\eta,j}, \sum_{j=1}^{n} z_{i,j} u_{\eta,j} = v_{\eta,i}, \tag{6}$$

i.e., the multiple sets of marginal distributions have a non-empty feasible region (see Appendix D for details). Multiple sets of marginal distributions could be jointly enforced by traversing the Sinkhorn iterations over k sets of marginal distributions. We extend Eq. (3) for multiple marginals,

$$\Gamma'^{(t)}_{i,j} = \frac{\Gamma^{(t)}_{i,j} v_{\eta,i}}{\sum_{j=1}^{n} \Gamma^{(t)}_{i,j} u_{\eta,j}}, \; \Gamma^{(t+1)}_{i,j} = \frac{\Gamma'^{(t)}_{i,j} u_{\eta,j}}{\sum_{i=1}^{m} \Gamma'^{(t)}_{i,j} u_{\eta,j}}, \tag{7}$$

where $\eta = (t \mod k) + 1$ is the index of marginal sets. Similarly to Algorithm 1, this generalized Sinkhorn algorithm finds a normalized matrix that is close to $\mathbf{S}$.

Theoretical Characterization of the Convergence of Multi-set Sinkhorn. In the following, we show that our proposed Algorithm 2 shares a similar convergence pattern with Algorithm 1 and Theorem 3.1. We generalize the theoretical steps in Chakrabarty & Khanna (2021) as follows.

We first study the convergence property of Algorithm 2 in terms of Kullback-Leibler (KL) divergence. In the following, we have $\eta = (t \mod k)+1$ unless otherwise specified. We define the probability over marginals $\pi_{v_{\eta,i}} = v_{\eta,i}/h_\eta$, and similarly for $\pi_{u_{\eta,j}}$. $\mathbf{v}^{(t)}_\eta, \mathbf{u}^{(t)}_\eta$ are the η-th marginal distributions achieved by $\mathbf{\Gamma}^{(t)}$ and $\mathbf{\Gamma}'^{(t)}$, respectively,

$$v^{(t)}_{\eta,i} = \sum_{j=1}^{n} \Gamma^{(t)}_{i,j} u_{\eta,j}, \; u^{(t)}_{\eta,j} = \sum_{i=1}^{m} \Gamma'^{(t)}_{i,j} u_{\eta,j}. \tag{8}$$

$D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}^{(t)}_\eta}) = \sum_{i=1}^{m} \frac{v_{\eta,i}}{h_\eta} \cdot \log \frac{v_{\eta,i}/h_\eta}{v^{(t)}_{\eta,i}/h_\eta}$ denotes the KL divergence between the current marginal achieved by $\mathbf{\Gamma}^{(t)}$ and the target marginal distribution. Similarly, we define $D_{\mathrm{KL}}(\pi_{\mathbf{u}_\eta} || \pi_{\mathbf{u}^{(t)}_\eta}) = \sum_{j=1}^{n} \frac{u_{\eta,j}}{h_\eta} \cdot \log \frac{u_{\eta,j}/h_\eta}{u^{(t)}_{\eta,j}/h_\eta}$ for $\mathbf{\Gamma}'^{(t)}$.

Theorem 3.2 (Converge Rate for KL divergence)**.** *For any $\delta > 0$, the Sinkhorn algorithm for multi-set*

marginals returns a matrix $\mathbf{\Gamma}^{(t)}$ or $\mathbf{\Gamma}'^{(t)}$ with KL divergence $\leq \delta$ in time $t = \mathcal{O}\left(\frac{k\log(\Delta/\alpha)}{\delta}\right)$, where $\alpha = \min_{i,j:s_{i,j}>0} s_{i,j}/\max_{i,j} s_{i,j}$, $\Delta = \max_j |\{i : s_{i,j} > 0\}|$ is the max number of non-zeros in any column of $\mathbf{S}$, and recall that k is the number of marginal sets.

Proof. (Sketch only and see Appendix B for details) To prove the upper bound of the convergence rate, we define the KL divergence for matrices $\mathbf{Z}, \mathbf{\Gamma}$ at η,

$$D(\mathbf{Z}, \mathbf{\Gamma}, \eta) = \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{z_{i,j}}{\Gamma_{i,j}}, \tag{9}$$

We then prove the convergence rate w.r.t. KL divergence based on the following two Lemmas.

Lemma 3.3. *For any η, $D(\mathbf{Z}, \mathbf{\Gamma}^{(0)}, \eta) \leq \log(1 + 2\Delta/\alpha)$.*

Lemma 3.4. *For $\eta = (t \mod k) + 1$, $\eta' = (t + 1 \mod k) + 1$, we have*

$$D(\mathbf{Z}, \mathbf{\Gamma}^{(t)}, \eta) - D(\mathbf{Z}, \mathbf{\Gamma}'^{(t)}, \eta) = D_{KL}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}})$$
$$D(\mathbf{Z}, \mathbf{\Gamma}'^{(t)}, \eta) - D(\mathbf{Z}, \mathbf{\Gamma}^{(t+1)}, \eta') = D_{KL}(\pi_{\mathbf{u}_\eta} || \pi_{\mathbf{u}_\eta^{(t)}})$$

The proof of these two Lemmas is referred to the appendix. Denote $T = \frac{k\log(1+2\Delta/\alpha)}{\delta} + \zeta$, where $\zeta \in [0, k)$ is a residual term ensuring $(T + 1) \mod k = 0$. If all $D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}}) > \delta$ and $D_{\mathrm{KL}}(\pi_{\mathbf{u}_\eta} || \pi_{\mathbf{u}_\eta^{(t)}}) > \delta$ for all η, by substituting Lemma 3.4 and summing, we have

$$D(\mathbf{Z}, \mathbf{\Gamma}^{(0)}, 1) - D(\mathbf{Z}, \mathbf{\Gamma}^{(T+1)}, 1) > \frac{T\delta}{k} \geq \log(1+2\Delta/\alpha).$$

As KL divergence is non-negative, the above formula contradicts to Lemma 3.3. It ends the proof of Theorem 3.2. □

For the L_1 error defined as

$$L_1(\mathbf{\Gamma}^{(t)}) = \|\mathbf{v}_\eta^{(t)} - \mathbf{v}_\eta\|_1,\ L_1(\mathbf{\Gamma}'^{(t)}) = \|\mathbf{u}_\eta^{(t)} - \mathbf{u}_\eta\|_1,$$

we have the following corollary for Algorithm 2,

Corollary 3.5 (Converge Rate for L_1-error)**.** *For any $\epsilon > 0$, the Sinkhorn algorithm for multi-set marginals returns a matrix $\mathbf{\Gamma}^{(t)}$ or $\mathbf{\Gamma}'^{(t)}$ with L_1 error $\leq \epsilon$ in time $t = \mathcal{O}\left(\frac{\hat{h}^2 k\log(\Delta/\alpha)}{\epsilon^2}\right)$ where $\hat{h} = \max_\eta \sum_{i=1}^{m} v_{\eta,i}$.*

Proof. Without loss of generality, we apply Pinsker's inequality $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q}) \geq \frac{1}{2}\|\mathbf{p} - \mathbf{q}\|_1^2$ to the row marginal distributions, and we have:

$$D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}}) \geq \frac{1}{2h_\eta^2} \|\mathbf{v}_\eta^{(t)} - \mathbf{v}_\eta\|_1^2 \tag{10a}$$
$$\geq \frac{1}{2\hat{h}^2} \|\mathbf{v}_\eta^{(t)} - \mathbf{v}_\eta\|_1^2. \tag{10b}$$

By setting $\delta = \frac{\epsilon^2}{2\hat{h}^2}$ in Theorem 3.2, in $t = \mathcal{O}\left(\frac{\hat{h}^2 k\log(\Delta/\alpha)}{\epsilon^2}\right)$ we have $\delta \geq D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}})$, corresponding to $\|\mathbf{v}_\eta^{(t)} - \mathbf{v}_\eta\|_1 \leq \epsilon$. Same conclusion holds for $\|\mathbf{u}_\eta^{(t)} - \mathbf{u}_\eta\|_1 \leq \epsilon$. □

Understanding the Underlying Formulation of Multi-set Sinkhorn. We further characterize the underlying formulation tackled by Algorithm 2 to better understand its behavior. Conceptually, Algorithm 2 finds a matrix $\mathbf{\Gamma}$ which is close to the non-negative score matrix $\mathbf{S}$. More formally, by introducing $\mathbf{W} \in \mathbb{R}^{m\times n}$ and $\mathbf{W} = \tau \log \mathbf{S}$, Sinkhorn for multi-set marginals tackles the following entropic regularized linear problem:

$$\min_{\mathbf{\Gamma}} \ -tr(\mathbf{W}^\top \mathbf{\Gamma}) + \tau \sum_{i,j} \Gamma_{i,j} \log \Gamma_{i,j}, \tag{11a}$$
$$s.t. \quad \mathbf{\Gamma} \in [0,1]^{m\times n}, \quad \forall \eta \in \{1, \cdots, k\}: \tag{11b}$$
$$\sum_{i=1}^{m} \Gamma_{i,j} u_{\eta,j} = u_{\eta,j}, \sum_{j=1}^{n} \Gamma_{i,j} u_{\eta,j} = v_{\eta,i}. \tag{11c}$$

where τ is the temperature hyperparameter for entropic regularization. Eq. (11) is tackled by first applying $\mathbf{S} = \exp(\mathbf{W}/\tau)$, and then calling Algorithm 2. When $\tau \to 0^+$, Eq. (11) degenerates to a linear programming problem. Since all constraints are positive linear and the objective is also linear, the solution to Eq. (11) usually lies at the vertices of the feasible space when $\tau \to 0^+$ i.e. most elements in $\mathbf{\Gamma}$ will be close to 0/1 given a small τ. Such a property is also in line with the "classic" single-set Sinkhorn (with entropic regularization): τ controls the "discreteness" of $\mathbf{\Gamma}$. A smaller τ makes the values in $\mathbf{\Gamma}$ closer to 0/1, and a larger τ leads to smoother $\mathbf{\Gamma}$. Please refer to Appendix C for more details about the entropic regularizer.

3.3. LinSAT: Enforcing Positive Linear Satisfiability

Denote $\mathbf{y}$ as an l-length vector that can be the output of any neural network. Our LinSAT develops an satisfiability layer that projects $\mathbf{y}$ into $\mathbf{x} \in [0,1]^l$, $\mathrm{LinSAT}(\mathbf{y}, \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d}, \mathbf{E}, \mathbf{f}) \to \mathbf{x}$, where $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{C}\mathbf{x} \geq \mathbf{d}, \mathbf{E}\mathbf{x} = \mathbf{f}$. $\mathbf{x}$ is dependent on $\mathbf{y}$ (following Eq. (11)) and, in the meantime, lies in the feasible space. We firstly show how to encode $\mathbf{y}$ and $\mathbf{x}$ by our proposed Algorithm 2.

Encoding Neural Network's Output. For an l-length vector denoted as $\mathbf{y}$, the following matrix is built

$$\mathbf{W} = \left[\begin{array}{cccc|c} y_1 & y_2 & \dots & y_l & \beta \\ \hline \beta & \beta & \dots & \beta & \beta \end{array}\right], \tag{12}$$

where $\mathbf{W}$ is of size $2 \times (l+1)$, and β is the dummy variable e.g. $\beta = 0$. $\mathbf{y}$ is put at the upper-left region of $\mathbf{W}$. The entropic regularizer is then enforced to control discreteness

and handle potential negative inputs:

$$\mathbf{S} = \exp\left(\frac{\mathbf{W}}{\tau}\right). \tag{13}$$

The score matrix $\mathbf{S}$ is taken as the input of Algorithm 2. LinSAT then enforces positive linear constraints to the corresponding region of $\mathbf{y}$ by regarding the constraints as marginal distributions.

From Linear Constraints to Marginal Distributions. We discuss the connections between positive linear constraints and marginal distributions for $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{C}\mathbf{x} \geq \mathbf{d}, \mathbf{E}\mathbf{x} = \mathbf{f}$, respectively. For notation's simplicity, here we discuss with only one constraint. Multiple constraints are jointly enforced by multiple sets of marginals.

Packing constraint $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Assuming that there is only one constraint, we rewrite the constraint as $\sum_{i=1}^{l} a_i x_i \leq b$. The marginal distributions are defined as

$$\mathbf{u}_p = \underbrace{[a_1 \quad a_2 \quad \dots \quad a_l \quad b]}_{l \text{ dims}+1 \text{ dummy dim}}, \ \mathbf{v}_p = \begin{bmatrix} b \\ \sum_{i=1}^{l} a_i \end{bmatrix}. \tag{14}$$

Following the "transportation" view of Sinkhorn (Cuturi, 2013), the output $\mathbf{x}$ *moves* at most b unit of mass from $a_1, a_2, \cdots, a_l$, and the dummy dimension allows the inequality by *moving* mass from the dummy dimension. It is also ensured that the sum of $\mathbf{u}_p$ equals the sum of $\mathbf{v}_p$.

Covering constraint $\mathbf{C}\mathbf{x} \geq \mathbf{d}$. Assuming that there is only one constraint, we rewrite the constraint as $\sum_{i=1}^{l} c_i x_i \geq d$. The marginal distributions are defined as

$$\mathbf{u}_c = \underbrace{[c_1 \quad c_2 \quad \dots \quad c_l \quad \gamma d]}_{l \text{ dims}+1 \text{ dummy dim}}, \ \mathbf{v}_c = \begin{bmatrix} (\gamma+1)d \\ \sum_{i=1}^{l} c_i - d \end{bmatrix}, \tag{15}$$

where the multiplier $\gamma = \left\lfloor \sum_{i=1}^{l} c_i / d \right\rfloor$ is necessary because we always have $\sum_{i=1}^{l} c_i \geq d$ (else the constraint is infeasible), and we cannot reach the feasible solution where all elements in $\mathbf{x}$ are 1s without this multiplier. This formulation ensures that at least d unit of mass is *moved* from $c_1, c_2, \cdots, c_l$ by $\mathbf{x}$, thus representing the covering constraint of "greater than". It is also ensured that the sum of $\mathbf{u}_c$ equals the sum of $\mathbf{v}_c$.

Equality constraint $\mathbf{E}\mathbf{x} = \mathbf{f}$. Representing the equality constraint is more straightforward. Assuming that there is only one constraint, we rewrite the constraint as $\sum_{i=1}^{l} e_i x_i = f$. The marginal distributions are defined as

$$\mathbf{u}_e = \underbrace{[e_1 \quad e_2 \quad \dots \quad e_l \quad 0]}_{l \text{ dims}+\text{dummy dim}=0}, \ \mathbf{v}_e = \begin{bmatrix} f \\ \sum_{i=1}^{l} e_i - f \end{bmatrix}, \tag{16}$$

where the output $\mathbf{x}$ *moves* $e_1, e_2, \cdots, e_l$ to f, and we need no dummy element in $\mathbf{u}_e$ because it is an equality constraint. It is also ensured that the sum of $\mathbf{u}_e$ equals the sum of $\mathbf{v}_e$.

Enforcing Multiple Constraints by Sinkhorn. The constraints are firstly modulated as multiple sets of marginals and then stacked into $\mathbf{U} \in \mathbb{R}_{\geq 0}^{k \times (l+1)}, \mathbf{V} \in \mathbb{R}_{\geq 0}^{k \times 2}$, where k is the number of constraints. By building $\mathbf{W}$ from $\mathbf{y}$, getting $\mathbf{S} = \exp(\mathbf{W}/\tau)$ and calling Algorithm 2 based on $\mathbf{S}, \mathbf{U}, \mathbf{V}$, the satisfiability of positive linear constraints is enforced to the output of neural networks.

Implementation Details. We set separate dummy variables for different constraints to handle potential conflicts among different sets of marginals (see explanations in Appendix D).

4. Case Study I: Neural Solver for Traveling Salesman Problem with Extra Constraints

4.1. Problem Background

The Traveling Salesman Problem (TSP) is a classic NP-hard problem. The standard TSP aims at finding a cycle visiting all cities with minimal length, and developing neural solvers for TSP receives increasing interest (Vinyals et al., 2015; Kool et al., 2019; Kwon et al., 2021). Beyond standard TSP, here we develop a neural solver for TSP with extra constraints using LinSAT layer.

4.2. Constraint Formulation for LinSAT

We consider 1) TSP with starting and ending cities constraint (TSP-SE); 2) TSP with priority constraint (TSP-PRI).

1) TSP-SE. We little abuse notations that appeared in Sec. 3. Given n cities and two of them are the starting and ending cities $s, e \in \{1, \dots, n\}$. The distance matrix $\mathbf{D} \in \mathbb{R}_{\geq 0}^{n \times n}$ records the distances between city pairs. TSP-SE finds the shortest tour starting from city s, visiting other cities exactly once, and ending in city e. TSP-SE can be formulated with the following objective function and constraints:

$$\min_{\mathbf{X}} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} D_{i,j} \sum_{k=1}^{n-1} X_{i,k} X_{j,k+1}, \tag{17}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} X_{i,k} = 1, \forall k \in \{1, \dots, n\}, \tag{17a}$$

$$\sum_{k=1}^{n} X_{i,k} = 1, \forall i \in \{1, \dots, n\}, \tag{17b}$$

$$X_{s,1} = 1, \quad X_{e,n} = 1, \tag{17c}$$

$$X_{i,k} \in \{0, 1\}, \quad \forall i, j \in \{1, \dots, n\}, \tag{17d}$$

where $\mathbf{X} \in \{0,1\}^{n \times n}$ is a binary matrix and $X_{i,k} = 1$ indicates city i is the k-th visited city in the tour. Constraints (17a) and (17b) ensure $\mathbf{X}$ to be a valid tour and

Table 2. Comparison of average tour length and total inference time for 10,000 testing instances on TSP variants with extra constraints. "Standard Solver" means state-of-the-art solvers for standard TSP. Our method is marked as gray.

	Method	TSP-SE		TSP-PRI	
		Tour Length	Time	Tour Length	Time
MIP	Gurobi (Sec.4.2, 2s)	4.608	5h34m	4.720	5h34m
	Gurobi (Sec.4.2, 10s)	4.010	27h44m	4.148	27h45m
Heuristic	Nearest Neighbor	4.367	0s	4.674	0s
	Nearest Insertion	4.070	9s	4.349	9s
	Farthest Insertion	3.772	11s	4.403	10s
	Random Insertion	3.853	5s	4.469	4s
Standard Solver	Gurobi (MTZ)	**3.648**	1h2m	-	-
	Concorde	**3.648**	9m28s	-	-
	LKH3	**3.648**	2m44s	-	-
Neural	Attention Model	3.677	4m39s	4.008	4m43s
	LinSAT (ours)	3.811	19s	**3.943**	18s

constraint (17c) defines the starting and ending cities. If $X_{i,k}X_{j,k+1} = 1$ for some k, then the k-th step of the tour is from i to j, and $D_{i,j}$ will be counted into the objective.

2) TSP-PRI. In practice, some cities may have higher priority and need to be visited earlier. In TSP-PRI we consider: in the given n cities, the priority city $p \neq s, e$ has to be visited within the first m steps. We add a new constraint to TSP-SE to formulate TSP-PRI:

$$\sum_{k=1}^{m+1} X_{p,k} = 1. \tag{18}$$

To fit with the continuous nature of neural networks, we relax the binary constraint (17d) to continuous ones $\widetilde{X}_{i,k} \in [0, 1]$ which is automatically satisfied by LinSAT. A neural network takes the instance as input and outputs the pre-projected matrix $\mathbf{Y} \in \mathbb{R}^{n\times n}$. $\mathbf{Y}$ is flattened into a n^2-dimensional vector and projected via LinSAT to enforce all the aforementioned constraints. Note that the neural network itself is a solver to an optimization problem, enforcing the constraint satisfiability by LinSAT is a reasonable choice instead of optimizing some other auxiliary objectives.

4.3. Network Design Details

Following the Attention Model for standard TSP (Kool et al., 2019), we use a Transformer (Vaswani et al., 2017) without positional encoding to encode each of the n nodes into a hidden vector $\mathbf{h}_i$. Learnable embeddings to mark starting, ending and priority cities are added to the corresponding embeddings before input to the Transformer. After encoding, $\mathbf{h}_i$ is projected into $\mathbf{Y}_i \in \mathbb{R}^n$ using an MLP. All $\mathbf{Y}_i, i \in \{1, \ldots, n\}$ form the pre-projected matrix $\mathbf{Y} \in \mathbb{R}^{n\times n}$. In training, the objective Eq. (17) with continuous $\widetilde{\mathbf{X}}$ as the decision variable is used as the unsupervised loss. For inference, we first output $\widetilde{\mathbf{X}}$. As $\widetilde{\mathbf{X}}$ satisfies constraints (17a) and (17b), it can be viewed as the marginal distributions of the binary $\mathbf{X}$ (Adams & Zemel, 2011). We perform beam search on $\widetilde{\mathbf{X}}$ to get $\mathbf{X}$ in post-processing.

4.4. Experiments

Following Kool et al. (2019), for both TSP variants, we generate 10,000 2-D Euclidean TSP instances as the testing set. Each instance consists of $n = 20$ nodes uniformly sampled in the unit square $[0, 1]^2$. The starting, ending, and priority cities are randomly selected. As our model is unsupervised, the training set is generated on the fly using the same process. For TSP-PRI, the number of priority steps is set to $m = 5$. The following baselines are considered with results shown in Table 2: 1) Mixed integer programming (MIP) solvers by directly applying the commercial solver Gurobi (Gurobi Optimization, 2020) to formulations in Sec. 4.2 and the time limit per instance is set as 2s/10s; 2) Heuristics e.g. nearest neighbor and insertion heuristics (Johnson, 1990) that are usually fast and approximate algorithms; 3) State-of-the-art solvers for standard TSP like Concorde[2] and LKH3 (Helsgaun, 2017), and Gurobi (MTZ) means applying Gurobi to the TSP formulation named after Miller-Tucker-Zemlin (MTZ) (Miller et al., 1960); 4) RL-based neural routing solver Attention Model (Kool et al., 2019).

Because the objective in Eq. (17) is quadratic w.r.t. $\mathbf{X}$, it is hard for a MIP solver to get a satisfactory solution quickly. Heuristic methods run much faster and perform well on TSP-SE, but their performances drop greatly on TSP-PRI. TSP-SE can be converted to the standard TSP, making it possible for standard solvers to find the optimal tour within a reasonable time. However, these highly specialized methods cannot be easily transferred to TSP-PRI. The RL-based Attention Model performs worse than our LinSAT on TSP-PRI as its performance highly depends on specialized decoding strategies for different tasks. Finally, our LinSAT can get

[2] https://www.math.uwaterloo.ca/tsp/concorde/index.html

near-optimal solutions in a short time on both TSP variants, and it is easy to transfer from TSP-SE to TSP-PRI by adding one single constraint. Details of TSP experiments are discussed in Appendix E.

5. Case Study II: Partial Graph Matching with Outliers on Both Sides

5.1. Problem Background

Standard graph matching (GM) assumes an outlier-free setting namely bijective mapping. One-shot GM neural networks (Wang et al., 2022) effectively enforce the satisfiability of one-to-one matching constraint by single-set Sinkhorn (Algorithm 1). Partial GM refers to the realistic case with outliers on both sides so that only a partial set of nodes are matched. There lacks a principled approach to enforce matching constraints for partial GM. The main challenge for existing GM networks is that they cannot discard outliers because the single-set Sinkhorn is outlier-agnostic and tends to match as many nodes as possible. The only exception is BBGM (Rolínek et al., 2020) which incorporates a traditional solver that can reject outliers, yet its performance still has room for improvement.

5.2. Constraint Formulation for LinSAT

Denote a graph pair by $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$, $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, where $|\mathcal{V}_1| = n_1$, $|\mathcal{V}_2| = n_2$. In mainstream GM networks, a matching score matrix $\mathbf{M} \in \mathbb{R}^{n_1 \times n_2}$ is expected to describe the correspondences of nodes between $\mathcal{G}_1$ and $\mathcal{G}_2$, where $\mathbf{M}_{i,j}$ refers to the matching score between node i in $\mathcal{G}_1$ and node j in $\mathcal{G}_2$. In previous bijective GM networks, the one-to-one node matching constraint that a node corresponds to at most one node is enforced by the off-the-shelf Sinkhorn algorithm in Algorithm 1. It cannot take the outliers into consideration, as it forcibly matches all nodes. The partial GM problem can be formulated by adding a partial matching constraint: assume that the number of inliers is ϕ, so the number of matched nodes should not exceed ϕ.

With a little abuse of notations, denote $\mathbf{X} \in [0,1]^{n_1 \times n_2}$ as the output of our partial GM network, the partial GM problem has the following constraints,

$$\sum_{i=1}^{n_1} X_{i,j} \leq 1, \forall j \in \{1, \dots, n_2\}, \tag{19a}$$

$$\sum_{j=1}^{n_2} X_{i,j} \leq 1, \forall i \in \{1, \dots, n_1\}, \tag{19b}$$

$$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} X_{i,j} \leq \phi. \tag{19c}$$

The constraint (19a) and (19b) denotes the node-matching on rows and columns, respectively, and they ensure (at most) one-to-one node correspondence. Constraint (19c) is the partial matching constraint ensuring that the total number of matched node pairs should not exceed ϕ. All constraints are positive linear and can be enforced by LinSAT layer. We implement our partial GM neural network by flattening $\mathbf{M}$ into a $n_1 n_2$-dimensional vector to feed into LinSAT.

Table 3. F1 (%) on Pascal VOC Keypoint (unfiltered setting). "Sinkhorn" denotes the classic single-set version in Algorithm 1.

GM Net	Constraint Technique	Matching Type	Mean F1
PCA-GM	Sinkhorn	bijective	48.6
BBGM	(Pogančić et al., 2019)	bijective	51.9
NGMv2	Sinkhorn	bijective	58.8
BBGM	(Pogančić et al., 2019)	partial	59.0
NGMv2	Sinkhorn+post-processing	partial	60.7
NGMv2	LinSAT (ours)	partial	**61.2**

5.3. Network Design Details

We follow the SOTA GM network NGMv2 (Wang et al., 2022) and replace the original Sinkhorn layer with LinSAT to tackle the partial GM problem on natural images. Specifically, a VGG16 (Simonyan & Zisserman, 2014) network is adopted to extract initial node features and global features from different CNN layers. The node features are then refined by SplineConv (Fey et al., 2018). The edge features are produced by the node features and the connectivity of graphs. The matching scores are predicted by the neural graph matching network proposed by Wang et al. (2022), finally generating the matching scores $\mathbf{M}$. We replace the original single-set Sinkhorn layer by LinSAT to enforce the constraints in Eq. (19). The output of LinSAT is reshaped into matrix $\hat{\mathbf{M}}$, which is used for end-to-end training with permutation loss (Wang et al., 2019b). During inference, the Hungarian algorithm (Kuhn, 1955) is performed on $\hat{\mathbf{M}}$ and we retain the ϕ-highest matching scores from $\hat{\mathbf{M}}$, and the remaining matches are discarded.

5.4. Experiments

We do experiments on Pascal VOC Keypoint dataset (Everingham et al., 2010) with Berkeley annotations (Bourdev & Malik, 2009) under the "unfiltered" setting following Rolínek et al. (2020) and report the matching F1 scores between graph pairs. We assume that the number of inliers ϕ is given (e.g. estimated by another regression model) and focus on the GM networks. As there are no one-shot partial GM networks available, we compare with bijective matching networks: PCA-GM (Wang et al., 2019b) and BBGM (Rolínek et al., 2020). We also build a partial GM baseline by post-processing retaining only the top-ϕ matches. Table 3 shows that our method performs the best. Note that BBGM (matching type=bijective) is an example of applying black-box solvers to an ill-posed optimization problem because the objective function does not consider the outliers, leading to

Table 4. Sharpe ratio of portfolio allocation methods. The constraint technique "Gurobi Opt" means solving a constrained optimization problem by the commercial solver Gurobi whereby the optimization parameters are based on the predicted asset prices.

Predictor	Constraint Technique	Expert Pref.?	Mean Sharpe
LSTM	Softmax	No	2.15
LSTM	Gurobi Opt	Yes	2.08
LSTM	LinSAT (ours)	Yes	**2.27**
StemGNN	Softmax	No	2.11
StemGNN	Gurobi Opt	Yes	2.00
StemGNN	LinSAT (ours)	Yes	**2.42**

inferior performance.

6. Case Study III: Portfolio Allocation

6.1. Problem Background

Predictive portfolio allocation is the process of selecting the best asset allocation based on predictions of future financial markets. The goal is to design an allocation plan to best trade-off between the return and the potential risk (i.e. the volatility). In an allocation plan, each asset is assigned a non-negative weight and all weights should sum to 1. Existing learning-based methods (Zhang et al., 2020; Butler & Kwon, 2021) only consider the sum-to-one constraint without introducing personal preference or expert knowledge. In contrast, we achieve such flexibility for the target portfolio via positive linear constraints: a mix of covering and equality constraints, which is widely considered (Sharpe, 1971; Mansini et al., 2014) for its real-world demand.

6.2. Constraint Formulation for LinSAT

Given historical data of assets, we aim to build a portfolio whose future Sharpe ratio (Sharpe, 1998) is maximized. Sharpe ratio $= \frac{\text{return}-\text{r}_\text{f}}{\text{risk}}$, where r_f denotes the risk-free return and is assumed to be 3% (annually). Besides the sum-to-one constraint, we consider the extra constraint based on expert preference: among all assets, the proportion of assets in set $\mathcal{C}$ should exceed p. This is reasonable as some assets (e.g. tech giants) have higher Sharpe ratios than others in certain time periods. Formally, the constraints are formulated as:

$$\sum_{i=1}^{n} x_i = 1, \quad \sum_{i \in \mathcal{C}} x_i \geq p, \tag{20}$$

where $\mathbf{x} \in [0,1]^n$ is the predicted portfolio. The first constraint is the traditional sum-to-one constraint and the second one is the extra preference constraint.

6.3. Network Design Details

We adopt LSTM (Hochreiter & Schmidhuber, 1997) and StemGNN (Cao et al., 2020) as two variants of portfolio allocation networks for their superiority in learning with time series. Our network has two output branches, one predicts future asset prices and the other predicts the portfolio. LinSAT is applied to the portfolio prediction branch to enforce constraints in Eq. (20). The network receives supervision signals by a weighted sum of maximizing the Sharpe ratio and minimizing the prediction error on future asset prices (based on the historical data in the training set).

6.4. Experiments

We consider the portfolio allocation problem where the network is given the historical data in the previous 120 trading days, and the goal is to build a portfolio with maximized Sharpe ratio for the next 120 trading days. The training set is built on the real prices of 494 assets from the S&P 500 index from 2018-01-01 to 2020-12-30, and the models are tested on real-world data from 2021-03-01 to 2021-12-30. Without loss of generality, we impose the expert preference that in the period of interest, the following tech giants' stocks could be more profitable: $\mathcal{C} = \{$*AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG*$\}$, and the preference ratio is set to $p = 50\%$.

We build two baselines: 1) A neural network portfolio allocator without preference, and the sum-to-one constraint is enforced by softmax following Zhang et al. (2020); 2) A two-stage allocator that first predicts future prices and then uses Gurobi (Gurobi Optimization, 2020) to solve a constrained optimization problem whose objective function is based on the predicted prices. See results in Table 4. Compared with an allocator without preference, the expert preference information improves the performance; Compared with the two-stage allocator, our allocator reduces the issue of error accumulation and builds better portfolios. Note that the objective function in the two-stage allocator is ill-posed because the first-stage prediction unavoidably contains errors.

7. Conclusion and Outlook

We have presented LinSAT, a principled approach to enforce the satisfiability of positive linear constraints for the solution as predicted in one-shot by neural network. The satisfiability layer is built upon an extended Sinkhorn algorithm for multi-set marginals, whose convergence is theoretically characterized. We showcase three applications of LinSAT. Future work may be improving the efficiency of both forward and backward of LinSAT.

Acknowledgments

The work was supported in part by National Key Research and Development Program of China (2020AAA0107600), NSFC (62222607, U19B2035), and Science and Technology Commission of Shanghai Municipality (22511105100).

References

Adams, R. and Zemel, R. Ranking via sinkhorn propagation. *arXiv:1106.1925*, 2011.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Allen-Zhu, Z. and Orecchia, L. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *Symp. on Disc. Algo.*, pp. 1439–1456, 2014.

Altschuler, J., Niles-Weed, J., and Rigollet, P. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *Neural Info. Process. Systems*, 30, 2017.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Int. Conf. Mach. Learn.*, pp. 136–145, 2017.

Awerbuch, B. and Khandekar, R. Stateless distributed gradient descent for positive linear programs. In *Symp. Theory of Comp.*, pp. 691–700, 2008.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *Eur. J. Operational Research*, 290(2):405–421, 2021.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable pertubed optimizers. *Neural Info. Process. Systems*, 33:9508–9519, 2020.

Bourdev, L. and Malik, J. Poselets: Body part detectors trained using 3d human pose annotations. In *Int. Conf. Comput. Vis.*, pp. 1365–1372, 2009.

Butler, A. and Kwon, R. Integrating prediction in mean-variance portfolio optimization. *Available at SSRN 3788875*, 2021.

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

Chakrabarty, D. and Khanna, S. Better and simpler error analysis of the sinkhorn–knopp algorithm for matrix scaling. *Mathematical Programming*, 188(1):395–407, 2021.

Cook, S. A. The complexity of theorem proving procedures. In *Symp. Theory of Comp.*, pp. 151—-158, 1971.

Cruz, S. R., Fernando, B., Cherian, A., and Gould, S. Deep-permnet: Visual permutation learning. *Comput. Vis. Pattern Recog.*, 2017.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. *Neural Info. Process. Systems*, pp. 2292–2300, 2013.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable ranking and sorting using optimal transport. In *Neural Info. Process. Systems*, volume 32, pp. 6858–6868, 2019.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.*, 2010.

Fey, M., Eric Lenssen, J., Weichert, F., and Müller, H. SplineCNN: Fast geometric deep learning with continuous b-spline kernels. In *Comput. Vis. Pattern Recog.*, pp. 869–877, 2018.

Guo, W., Zhen, H.-L., Li, X., Yuan, M., Jin, Y., and Yan, J. Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, 2023.

Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com, 2020.

Helsgaun, K. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Johnson, D. S. Local optimization and the traveling salesman problem. In *International colloquium on automata, languages, and programming*, pp. 446–461, 1990.

Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Karalias, N. and Loukas, A. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Neural Info. Process. Systems*, 2020.

Knight, P. A. The sinkhorn–knopp algorithm: convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *Int. Conf. Learn. Rep.*, pp. 1–25, 2019.

Kuhn, H. W. The hungarian method for the assignment problem. In *Export. Naval Research Logistics Quarterly*, pp. 83–97, 1955.

Kwon, Y.-D., Choo, J., Yoon, I., Park, M., Park, D., and Gwon, Y. Matrix encoding networks for neural combinatorial optimization. In *Neural Info. Process. Systems*, volume 34, pp. 5138–5149, 2021.

Li, Y., Chen, X., Guo, W., Li, X., Luo, W., Huang, J., Zhen, H.-L., Yuan, M., and Yan, J. Hardsatgen: Understanding the difficulty of hard sat formula generation and a strong structure-hardness-aware baseline. In *SIGKDD Conf. on Know. Disc. and Data Mining*, 2023.

Liu, C., Wang, R., Jiang, Z., Huang, L., Lu, P., and Yan, J. Revocable deep reinforcement learning with affinity regularization for outlier-robust graph matching. In *Int. Conf. Learn. Rep.*, 2023.

Luby, M. and Nisan, N. A parallel approximation algorithm for positive linear programming. In *Symp. Theory of Comp.*, pp. 448–457, 1993.

Mansini, R., Ogryczak, W., and Speranza, M. G. Twenty years of linear programming based portfolio optimization. *Eur. J. Operational Research*, 234(2):518–535, 2014.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 1960.

Ozolins, E., Freivalds, K., Draguns, A., Gaile, E., Zakovskis, R., and Kozlovics, S. Goal-aware neural sat solver. *arXiv preprint arXiv:2106.07162*, 2021.

Pass, B. Multi-marginal optimal transport: theory and applications. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 49(6):1771–1790, 2015.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Paulus, A., Rolínek, M., Musil, V., Amos, B., and Martius, G. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *Int. Conf. Mach. Learn.*, pp. 8443–8453, 2021.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of blackbox combinatorial solvers. In *Int. Conf. Learn. Rep.*, 2019.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. Deep graph matching via blackbox differentiation of combinatorial solvers. In *Eur. Conf. Comput. Vis.*, pp. 407–424, 2020.

Sason, I. and Verdú, S. Upper bounds on the relative entropy and rényi divergence as a function of total variation distance for finite alphabets. In *IEEE Information Theory Workshop-Fall*, pp. 214–218, 2015.

Selsam, D., Lamm, M., Bunz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a sat solver from single-bit supervision. In *Int. Conf. Learn. Rep.*, 2019.

Sharpe, W. F. A linear programming approximation for the general portfolio analysis problem. *Journal of Financial and Quantitative Analysis*, 6(5):1263–1275, 1971.

Sharpe, W. F. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, pp. 169–185, 1998.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Int. Conf. Learn. Rep.*, 2014.

Sinkhorn, R. and Knopp, P. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Neural Info. Process. Systems*, pp. 2692–2700, 2015.

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Int. Conf. Mach. Learn.*, pp. 6545–6554, 2019a.

Wang, R., Yan, J., and Yang, X. Learning combinatorial embedding networks for deep graph matching. In *Int. Conf. Comput. Vis.*, pp. 3056–3065, 2019b.

Wang, R., Yan, J., and Yang, X. Combinatorial learning of robust deep graph matching: an embedding based approach. *Trans. Pattern Anal. Mach. Intell.*, 45(6):6984–7000, 2020.

Wang, R., Hua, Z., Liu, G., Zhang, J., Yan, J., Qi, F., Yang, S., Zhou, J., and Yang, X. A bi-level framework for learning to solve combinatorial optimization on graphs. In *Neural Info. Process. Systems*, 2021a.

Wang, R., Zhang, T., Yu, T., Yan, J., and Yang, X. Combinatorial learning of graph edit distance via dynamic embedding. *Comput. Vis. Pattern Recog.*, 2021b.

Wang, R., Yan, J., and Yang, X. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *Trans. Pattern Anal. Mach. Intell.*, 44(9):5261–5279, 2022.

Wang, R., Shen, L., Chen, Y., Yang, X., Tao, D., and Yan, J. Towards one-shot neural combinatorial optimization solvers: Theoretical and empirical notes on the cardinality-constrained case. In *Int. Conf. Learn. Rep.*, 2023.

Xie, Y., Dai, H., Chen, M., Dai, B., Zhao, T., Zha, H., Wei, W., and Pfister, T. Differentiable top-k with optimal transport. In *Neural Info. Process. Systems*, volume 33, pp. 20520–20531, 2020.

Yan, J., Yang, S., and Hancock, E. R. Learning for graph matching and related combinatorial optimization problems. In *Int. Joint Conf. Artificial Intell.*, 2020.

Young, N. E. Sequential and parallel algorithms for mixed packing and covering. In *Found. of Comp. Sci.*, pp. 538–546, 2001.

Yu, T., Wang, R., Yan, J., and Li, B. Learning deep graph matching with channel-independent embedding and hungarian attention. In *Int. Conf. Learn. Rep.*, 2020.

Zhang, Z., Zohren, S., and Roberts, S. Deep learning for portfolio optimization. *The Journal of Financial Data Science*, 2(4):8–20, 2020.

A. Comparison with the Notations from Cuturi (2013)

The formulation used in this paper (regarding $\mathbf{\Gamma}$) is an equivalent adaptation from the notations used in existing single-set Sinkhorn papers e.g. Cuturi (2013). As we explained in the footnote in page 3, this new formulation is preferred as we are generalizing the scope of Sinkhorn to multi-set marginal, and the existing formulation cannot seamlessly handle marginals with different values.

Specifically, we make a side-by-side comparison with the notations used in this paper and the notations used in Cuturi (2013) on single-set Sinkhorn algorithm:

- This paper's notations:

  The transportation matrix is $\mathbf{\Gamma} \in [0,1]^{m\times n}$, and the constraints are

$$\sum_{i=1}^{m} \Gamma_{i,j} u_j = u_j, \quad \sum_{j=1}^{n} \Gamma_{i,j} u_j = v_i. \tag{21}$$

  $\Gamma_{i,j}$ means the *proportion* of u_j moved from u_j to v_i.

  The algorithm steps are:

  repeat:

  $\Gamma'_{i,j} = \frac{\Gamma_{i,j} v_i}{\sum_{j=1}^{n} \Gamma_{i,j} u_j}$; ▷ normalize w.r.t. $\mathbf{v}$

  $\Gamma_{i,j} = \frac{\Gamma'_{i,j} u_j}{\sum_{i=1}^{m} \Gamma'_{i,j} u_j}$; ▷ normalize w.r.t. $\mathbf{u}$

  until convergence.

- Cuturi (2013)'s notations:

  The transportation matrix is $\mathbf{P} \in \mathbb{R}_{\geq 0}^{m\times n}$, and the constraints are

$$\sum_{i=1}^{m} P_{i,j} = u_j, \quad \sum_{j=1}^{n} P_{i,j} = v_i. \tag{22}$$

  $P_{i,j}$ means the *exact mass* moved from u_j to v_i.

  The algorithm steps are:

  repeat:

  $P'_{i,j} = \frac{P_{i,j} v_i}{\sum_{j=1}^{n} P_{i,j}}$; ▷ normalize w.r.t. $\mathbf{v}$

  $P_{i,j} = \frac{P'_{i,j} u_j}{\sum_{i=1}^{m} P'_{i,j}}$; ▷ normalize w.r.t. $\mathbf{u}$

  until convergence.

The equivalence between the above formulations becomes clear if we substitute $P_{i,j}$ by $\Gamma_{i,j} u_j$ and $P'_{i,j}$ by $\Gamma'_{i,j} u_j$ in all the above definitions and algorithm steps. We would like to highlight that such different notations are necessary because making $\Gamma_{i,j}$ as the proportion of all $u_{1,j}, u_{2,j}, \cdots, u_{k,j}$ could deal with multiple marginals with different values, making it generalizable to the multi-marginal case.

B. Proof of Theorem 3.2.

To prove the upper bound of convergence rate, we define the Kullback-Leibler (KL) divergence for matrices $\mathbf{Z}$ and $\mathbf{\Gamma}$, whereby the KL divergence is originally defined for probability vectors,

$$D(\mathbf{Z}, \mathbf{\Gamma}, \eta) = \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{z_{i,j}}{\Gamma_{i,j}}. \tag{23}$$

Also recall that $\mathbf{Z} \in [0,1]^{m\times n}$ is a normalized matrix satisfying all marginal distributions,

$$\forall \eta \in \{1,\cdots,k\} : \sum_{i=1}^{m} z_{i,j}u_{\eta,j} = u_{\eta,j}, \sum_{j=1}^{n} z_{i,j}u_{\eta,j} = v_{\eta,i}, \tag{24}$$

We then prove the convergence rate w.r.t. the KL divergence based on the following two Lemmas.

Lemma 3.3. For any $\eta = 1,\cdots,k$, $D(\mathbf{Z},\mathbf{\Gamma}^{(0)},\eta) \le \log(1+2\Delta/\alpha)$.

Proof. By definition,

$$D(\mathbf{Z},\mathbf{\Gamma}^{(0)},\eta) = \frac{1}{h_\eta}\sum_{j=1}^{n} u_{\eta,j}\sum_{i=1}^{m} z_{i,j}\log\frac{z_{i,j}}{\Gamma_{i,j}}, \tag{25}$$

where $\{z_{i,j}\}_{i=1,2,\cdots,m}$ and $\{\Gamma_{i,j}\}_{i=1,2,\cdots,m}$ sum to 1 thus they are probability distributions. The second summand is a standard KL divergence term.

Based on the following fact in terms of Eq. (27) from Sason & Verdú (2015), for probability distributions $\mathbf{p},\mathbf{q}$,

$$D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q}) \le \log\left(1+\frac{\|\mathbf{p}-\mathbf{q}\|_2^2}{q_{min}}\right) \tag{26}$$

$$\le \log\left(1+\frac{2}{q_{min}}\right), \tag{27}$$

where Eq. (26) to Eq. (27) is because $\|\mathbf{p}-\mathbf{q}\|_2 \le \sqrt{2}$, and q_{min} denotes the smallest non-zero element in $\mathbf{q}$. We have the following conclusion for $\mathbf{Z}$ and $\mathbf{\Gamma}^{(0)}$,

$$\begin{aligned} D(\mathbf{Z},\mathbf{\Gamma}^{(0)},\eta) &\le \frac{1}{h_\eta}\sum_{j=1}^{n} u_{\eta,j}\log\left(1+\frac{2}{\Gamma^{(0)}_{min}}\right) \\ &= \log\left(1+\frac{2}{\Gamma^{(0)}_{min}}\right), \end{aligned} \tag{28}$$

where $\frac{1}{h_\eta}\sum_{j=1}^{n} u_{\eta,j} = 1$ by definition.

Recall that $\alpha = \min_{i,j:s_{i,j}>0} s_{i,j}/\max_{i,j} s_{i,j}$, $\Delta = \max_j |\{i : s_{i,j} > 0\}|$ is the max number of non-zeros in any column of $\mathbf{S}$, and in our algorithm $\Gamma^{(0)}_{i,j} = \frac{s_{i,j}}{\sum_{i=1}^{m} s_{i,j}}$, we have

$$\Gamma^{(0)}_{min} \ge \frac{\alpha}{\Delta} \tag{29}$$

$$\Rightarrow \quad D(\mathbf{Z},\mathbf{\Gamma}^{(0)},\eta) \le \log\left(1+\frac{2\Delta}{\alpha}\right). \tag{30}$$

This ends the proof of Lemma 3.3. □

Lemma 3.4. For $\eta = (t \mod k)+1$, $\eta' = (t+1 \mod k)+1$, we have

$$D(\mathbf{Z},\mathbf{\Gamma}^{(t)},\eta) - D(\mathbf{Z},\mathbf{\Gamma}'^{(t)},\eta) = D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta}||\pi_{\mathbf{v}_\eta^{(t)}}) \tag{31}$$

$$D(\mathbf{Z},\mathbf{\Gamma}'^{(t)},\eta) - D(\mathbf{Z},\mathbf{\Gamma}^{(t+1)},\eta') = D_{\mathrm{KL}}(\pi_{\mathbf{u}_\eta}||\pi_{\mathbf{u}_\eta^{(t)}}) \tag{32}$$

Proof. By definition, the left-hand side of the first inequality is

$$\begin{aligned}
D(\mathbf{Z}, \mathbf{\Gamma}^{(t)}, \eta) - D(\mathbf{Z}, \mathbf{\Gamma}'^{(t)}, \eta) &= \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{z_{i,j}}{\Gamma_{i,j}^{(t)}} - \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{z_{i,j}}{\Gamma_{i,j}'^{(t)}} \\
&= \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{\Gamma_{i,j}'^{(t)}}{\Gamma_{i,j}^{(t)}} \\
&= \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{v_{\eta,i}}{\sum_{j=1}^{n} \Gamma_{i,j}^{(t)} u_{\eta,j}} \\
&= \frac{1}{h_\eta} \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \log \frac{v_{\eta,i}}{v_{\eta,i}^{(t)}} \\
&= \frac{1}{h_\eta} \sum_{i=1}^{m} \log \frac{v_{\eta,i}}{v_{\eta,i}^{(t)}} \sum_{j=1}^{n} z_{i,j} u_{\eta,j} \\
&= \sum_{i=1}^{m} \frac{v_{\eta,i}}{h_\eta} \log \frac{v_{\eta,i}/h_\eta}{v_{\eta,i}^{(t)}/h_\eta}
\end{aligned} \tag{33}$$

which is exactly $D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}})$. The other equation could be derived analogously. This ends the proof of Lemma 3.4. □

Denote $T = \frac{k \log(1+2\Delta/\alpha)}{\delta} + \zeta$, where $\zeta \in [0, k)$ is a residual term ensuring $(T+1) \mod k = 0$. If all $D_{\mathrm{KL}}(\pi_{\mathbf{v}_\eta} || \pi_{\mathbf{v}_\eta^{(t)}}) > \delta$ and $D_{\mathrm{KL}}(\pi_{\mathbf{u}_\eta} || \pi_{\mathbf{u}_\eta^{(t)}}) > \delta$ for all η, by substituting Lemma 3.4 and summing, we have

$$D(\mathbf{Z}, \mathbf{\Gamma}^{(0)}, 1) - D(\mathbf{Z}, \mathbf{\Gamma}^{(T+1)}, 1) > \frac{T\delta}{k} \geq \log(1 + 2\Delta/\alpha). \tag{34}$$

The multiplier k exists because we need to sum over k sets of marginals to cancel all intermediate terms. Since KL divergence must be non-negative, the above formula contradicts with Lemma 3.3. This ends the proof of Theorem 3.2. □

C. Further Discussions with the Entropic Regularizer

In the main paper, we start the formulation after the regularization term for simplicity. On one hand, the entropic regularizer may be omitted if the score matrix is non-negative (e.g. activated by ReLU). On the other hand, our theoretical insights could naturally generalize with the regularizer. We provide the detailed discussions as follows.

C.1. The Underlying Formulation of Algorithm 2

Recall that given real-valued matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, regularizer τ, and a set of target marginals $\mathbf{u}_\eta \in \mathbb{R}_{\geq 0}^n$, $\mathbf{v}_\eta \in \mathbb{R}_{\geq 0}^m$, our multi-set marginal Sinkhorn algorithm maps $\mathbf{W}$ to $\mathbf{\Gamma} \in [0,1]^{m \times n}$ such that $\forall \eta \in \{1, \cdots, k\} : \sum_{i=1}^{m} \Gamma_{i,j} u_{\eta,j} = u_{\eta,j}, \sum_{j=1}^{n} \Gamma_{i,j} u_{\eta,j} = v_{\eta,i}$.

Formally, the following entropic regularized problem is considered:

$$\min_{\mathbf{\Gamma}} \; -tr(\mathbf{W}^\top \mathbf{\Gamma}) + \tau \sum_{i,j} \Gamma_{i,j} \log \Gamma_{i,j}, \tag{35a}$$

$$s.t. \quad \mathbf{\Gamma} \in [0,1]^{m \times n}, \tag{35b}$$

$$\forall \eta \in \{1, \cdots, k\} : \sum_{i=1}^{m} \Gamma_{i,j} u_{\eta,j} = u_{\eta,j}, \sum_{j=1}^{n} \Gamma_{i,j} u_{\eta,j} = v_{\eta,i}. \tag{35c}$$

With dual variables $\theta_{\eta,u} \in \mathbb{R}^n, \theta_{\eta,v} \in \mathbb{R}^m$, the Lagrangian of the above problem is

$$\mathcal{L}(\Gamma, \theta_*) = \sum_{i,j} \tau \Gamma_{i,j} \log \Gamma_{i,j} - w_{i,j} \Gamma_{i,j} + \sum_{\eta=1}^{k} \theta_{\eta,v}^\top (\mathbf{\Gamma} \mathbf{u}_\eta - \mathbf{v}_\eta) + \theta_{\eta,u}^\top (\mathbf{\Gamma}^\top \mathbf{1}_m - \mathbf{1}_n), \tag{36}$$

and for any (i, j),

$$\frac{\partial \mathcal{L}}{\partial \Gamma_{i,j}} = 0 \tag{37a}$$

$$\Rightarrow \quad \tau \log \Gamma_{i,j} + \tau - w_{i,j} + \sum_{\eta=1}^{k} \theta_{\eta,v,i} u_{\eta,j} + \theta_{\eta,u,j} = 0 \tag{37b}$$

$$\Rightarrow \quad \Gamma_{i,j} = \exp(-\frac{1}{2} - \frac{1}{\tau} \sum_{\eta=1}^{k} \theta_{\eta,v,i} u_{\eta,j}) \exp(\frac{w_{i,j}}{\tau}) \exp(-\frac{1}{2} - \frac{1}{\tau} \sum_{\eta=1}^{k} \theta_{\eta,u,j}) \tag{37c}$$

Theorem 3.2 and Corollary 3.5 derives the convergence speed of transforming $\mathbf{S} = \exp(\mathbf{W}/\tau)$ into $\mathbf{\Gamma}$ by alternative row-wise and column-wise normalization. In other words, Sinkhorn's theorem (Sinkhorn & Knopp, 1967) could be generalized to the multi-set case, and the convergence speed is already derived in Theorem 3.2 and Corollary 3.5. $\mathbf{\Gamma}$ is necessarily a matrix of the form $\mathrm{diag}(\mathbf{a})\mathbf{S}\mathrm{diag}(\mathbf{b})$ that belongs to all marginals.

C.2. The Algorithm with Entropic Regularizer

Algorithm 3 Sinkhorn for Multi-Set Marginals with Entropic Regularizer

Input: Score matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, entropic regularizer τ, k sets of marginals $\mathbf{V} \in \mathbb{R}_{\geq 0}^{k \times m}$, $\mathbf{U} \in \mathbb{R}_{\geq 0}^{k \times n}$.
Apply entropic regularizer $\mathbf{S} = \exp(\mathbf{W}/\tau)$;
Initialize $\Gamma_{i,j} = \frac{s_{i,j}}{\sum_{i=1}^{m} s_{i,j}}$;
repeat
  for $\eta = 1$ **to** k **do**
    $\Gamma'_{i,j} = \frac{\Gamma_{i,j} v_{\eta,i}}{\sum_{j=1}^{n} \Gamma_{i,j} u_{\eta,j}}$; ▷ normalize w.r.t. $\mathbf{v}_\eta$
    $\Gamma_{i,j} = \frac{\Gamma'_{i,j} u_{\eta,j}}{\sum_{i=1}^{m} \Gamma'_{i,j} u_{\eta,j}}$; ▷ normalize w.r.t. $\mathbf{u}_\eta$
  end for
until convergence

C.3. Theoretical Results with Entropic Regularizer

If the entropic regularizer is involved, the converging rate of multi-set Sinkhorn w.r.t. L_1 error becomes:

$$t = \mathcal{O}\left(\frac{\hat{h}^2 k(\alpha'/\tau + \log \Delta)}{\epsilon^2}\right), \tag{38}$$

where

$$\alpha' = \max_{i,j} w_{i,j} - \min_{i,j: w_{i,j} > -\infty} w_{i,j}.$$

The other notations have the same definition as in the main paper. It shows that the number of iterations scales almost linearly with $1/\tau$. The derivation is straightforward since α is the only affected term after considering the entropic regularizer.

C.4. Empirical Further Study of the Entropic Regularizer

Our LinSAT can naturally handle continuous constraints. For continuous optimization problems (e.g. portfolio allocation) the output can be directly used as the feasible solution. When it comes to problems requiring discrete decision variables, we show in the following study that our LinSAT still owns the ability to encode the constraints by adjusting τ.

For discrete optimization problems (such as TSP and GM), during training, we relax the discrete binary region $\{0, 1\}$ into continuous $[0, 1]$ to make neural networks trainable with gradient-based methods. In Sections 4 and 5, the discrete solutions from continuous outputs are recovered via post-processing during inference. Post-processing is quite common in neural solvers for discrete optimization due to the continuous nature of neural networks. Let's take TSP as an example: RL-based

Table 5. Feasible ratio and tour length comparison among different TSP solver configurations.

Config	TSP-SE Feasible Ratio	TSP-SE Tour Length	TSP-PRI Feasible Ratio	TSP-PRI Tour Length
$\tau = 0.1$, Rounding	1.19%	3.897	1.52%	3.997
$\tau = 0.05$, Rounding	20.95%	3.904	23.26%	4.063
$\tau = 0.01$, Rounding	86.69%	3.964	85.32%	4.102
$\tau = 0.005$, Rounding	89.35%	3.969	83.63%	4.108
$\tau = 0.1$, Beam Search	100.00%	3.811	100.00%	3.943

Table 6. Feasible ratio and average F1 among different partial-GM solver configurations.

Config	Feasible Ratio	Mean F1
$\tau = 0.1$, Rounding	56.78%	0.4756
$\tau = 0.05$, Rounding	83,99%	0.5547
$\tau = 0.02$, Rounding	95.21%	0.5748
$\tau = 0.01$, Rounding	96.20%	0.5673
$\tau = 0.05$, Hungarian-Top-ϕ	100.00%	0.6118

methods (Kool et al., 2019; Kwon et al., 2021) need to manually design the decoding strategy to ensure every node is visited but only once; supervised learning methods (Joshi et al., 2019) output a continuous heatmap over which search is performed.

As for our LinSAT, once the network is well-trained, we can set smaller τ to get outputs that are closer to a discrete feasible solution during inference. Here we conduct experiments on TSP and GM to evaluate the LinSAT's capability to maintain discrete feasible solutions. Using the same trained network, we adjust τ to get different continuous outputs $\widetilde{X}$. Then we directly round $\widetilde{X}$ to get discrete solution X: $X_{i,j} = 1$ if $\widetilde{X}_{i,j} \geq 0.5$; $X_{i,j} = 0$ if $\widetilde{X}_{i,j} < 0.5$. In Tables 5 and 6, we report the ratio that directly rounded solutions are feasible and the corresponding evaluation metrics for feasible solutions.

Results show that our LinSAT can recover most feasible solutions with a simple rounding strategy under a small temperature. However, smaller τ requires more iterations to converge, which makes inference slower: in TSP-SE, $\tau = 0.005$ requires 1,000 iterations to converge and the total inference time for $\tau = 0.005$, Rounding is 37s, which is longer than 19s of $\tau = 0.1$, Beam Search. And the quality of solutions is also not competitive with larger τ with post-processing.

In summary, the aforementioned further study shows that: 1) Given smaller τ, our LinSAT owns the ability to recover discrete feasible solutions directly. 2) Using a larger τ with post-processing steps (as done in our main paper) is a cost-efficient choice considering both efficiency and efficacy.

D. The Feasibility Assumption Explained

For multi-set Sinkhorn, an assumption is made in Eq. (6) that the marginal distributions must have a non-empty feasible region. We explain this assumption with an example derived from positive linear constraints.

As shown in Section 3.3, every set of positive linear constraints could be equivalently viewed as a set of Sinkhorn marginals. If we transform the positive linear constraints to Sinkhorn's marginals:

- If the positive linear constraints have a non-empty feasible region, e.g.

$$x_1 + x_2 \leq 1, x_3 + x_4 \leq 1, x_1 + x_3 \leq 1, x_2 + x_4 \leq 1, \tag{39}$$

the corresponding marginals are

$$\mathbf{u}_1 = [1,1,0,0,1], \mathbf{v}_1^\top = [1,2] \quad \triangleright \text{for } x_1 + x_2 \leq 1 \tag{40a}$$
$$\mathbf{u}_2 = [0,0,1,1,1], \mathbf{v}_2^\top = [1,2] \quad \triangleright \text{for } x_3 + x_4 \leq 1 \tag{40b}$$
$$\mathbf{u}_3 = [1,0,1,0,1], \mathbf{v}_3^\top = [1,2] \quad \triangleright \text{for } x_1 + x_3 \leq 1 \tag{40c}$$
$$\mathbf{u}_4 = [0,1,0,1,1], \mathbf{v}_4^\top = [1,2] \quad \triangleright \text{for } x_2 + x_4 \leq 1 \tag{40d}$$
$$\tag{40e}$$

A feasible solution $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$ corresponds to a valid transportation plan,

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{41}$$

Thus Eq. (6) is feasible in such cases.

- If the positive linear constraints have no feasible region, e.g.

$$x_1 + x_2 \geq 2, x_3 + x_4 \geq 2, x_1 + x_3 \leq 1, x_2 + x_4 \leq 1, \tag{42}$$

Eq. (6) is infeasible. Such infeasible marginals are not expected, and LinSAT will not converge.

As a side note, the last column and the second row of $\mathbf{\Gamma}$ are set as separate dummy columns/rows for different marginals in our LinSAT, and their converged values may be different for different marginals. Empirically, LinSAT's converging property is not affected after adding the dummy columns/rows.

E. Details of Case Study I: Neural Solver for Traveling Salesman Problem with Extra Constraints

E.1. Hyper-parameters and Implementation

In both TSP-SE and TSP-PRI, we use a 3-layer Transformer to encode the 2-D coordinates into hidden vectors. Then the hidden vectors are projected into the pre-projected matrix using a 3-layer MLP with ReLU activation. Dimensions of hidden states for both the Transformer and the MLP are set to 256, and the head number of multi-head attention in the Transformer is set to 8.

We train the model for 100 epochs for both TSP-SE and TSP-PRI. In each epoch, we randomly generate 256,000 instances as the training set of this epoch. The batch size is set to 1,024. Adam optimizer is used for training and the learning rate is set to 1e-4.

During inference, we use beam search to post-process the continuous matrix $\widetilde{\mathbf{X}}$ output by the network to get the binary matrix $\mathbf{X}$. The width of the beam for beam search is set to 2,048.

Our model runs on a single NVIDIA GeForce RTX 2080Ti GPU with 11GB memory.

E.2. Baseline Methods

MIP MIP methods directly use Gurobi to solve the formulation in Sec.4.2, e.g. an integer programming problem with linear constraints and quadratic objective. The time limit per instance is set to 2s/20s.

Nearest Neighbor Nearest Neighbor is a greedy heuristic for TSP. For TSP-SE, in each iteration, the nearest node (except the ending node) to the starting node is selected as the next node to visit. Then the selected node becomes the new starting node in the next iteration. After all nodes except the ending node are visited, the tour directly connects to the ending node.

For TSP-PRI, in the m-th iteration, if the priority node has not been visited, the priority node will be selected as the next node to satisfy the priority constraint.

Table 7. Ablation study on normalizing randomly generated matrices on TSP.

Method	TSP-SE	TSP-PRI
Random Pre-Projected Matrix	7.414 (19s)	7.426 (18s)
Trainable Pre-Projected Matrix	5.546 (26m51s)	5.646 (27m37s)
Transformer Feature Matrix	**3.811** (19s)	**3.943** (18s)

Insertion Heuristic Insertion Heuristic first uses the starting and ending nodes to construct a partial tour. In each iteration, a new node is selected and inserted to the partial tour to extend it. For a selected node, it is inserted in the position where the tour length increase is minimized. Formally, we use $T = \{\pi_1, \pi_2, \ldots, \pi_m\}$ to denote a partial tour with $m(m < n)$ nodes. Assuming the selected new node is u^*, then it is inserted behind the i^*-th node in the partial tour:

$$i^* = \underset{1 \leq i \leq m-1}{\operatorname{argmin}} \; D_{\pi_i, u^*} + D_{u^*, \pi_{i+1}} - D_{\pi_i, \pi_{i+1}} \tag{43}$$

According to the different new node selection processes, there are different variants of insertion heuristic:

Nearest Insertion selects the nearest node to the partial tour:

$$u^* = \underset{u \notin T}{\operatorname{argmin}} \min_{v \in T} D_{u,v} \tag{44}$$

Farthest Insertion selects the farthest node from any node in the partial tour:

$$u^* = \underset{u \notin T}{\operatorname{argmax}} \min_{v \in T} D_{u,v} \tag{45}$$

Random Insertion randomly selects the new node to insert.

To satisfy the priority constraint of TSP-PRI, if the priority node is already the $(m+1)$-th node in the partial tour, we can not insert new nodes in front of it.

Standard Solver TSP-SE can be converted to standard TSP by adding a dummy node. The distance from the dummy node to starting and ending nodes is 0 and the distance to other nodes is infinity. Then we can use start-of-the-art methods for standard TSP, i.e. *Gurobi(MTZ)/Concrode/LKH3*, to solve TSP-SE.

However, converting TSP-PRI to standard TSP is non-trivial, making it hard to use standard solvers to solve TSP-PRI.

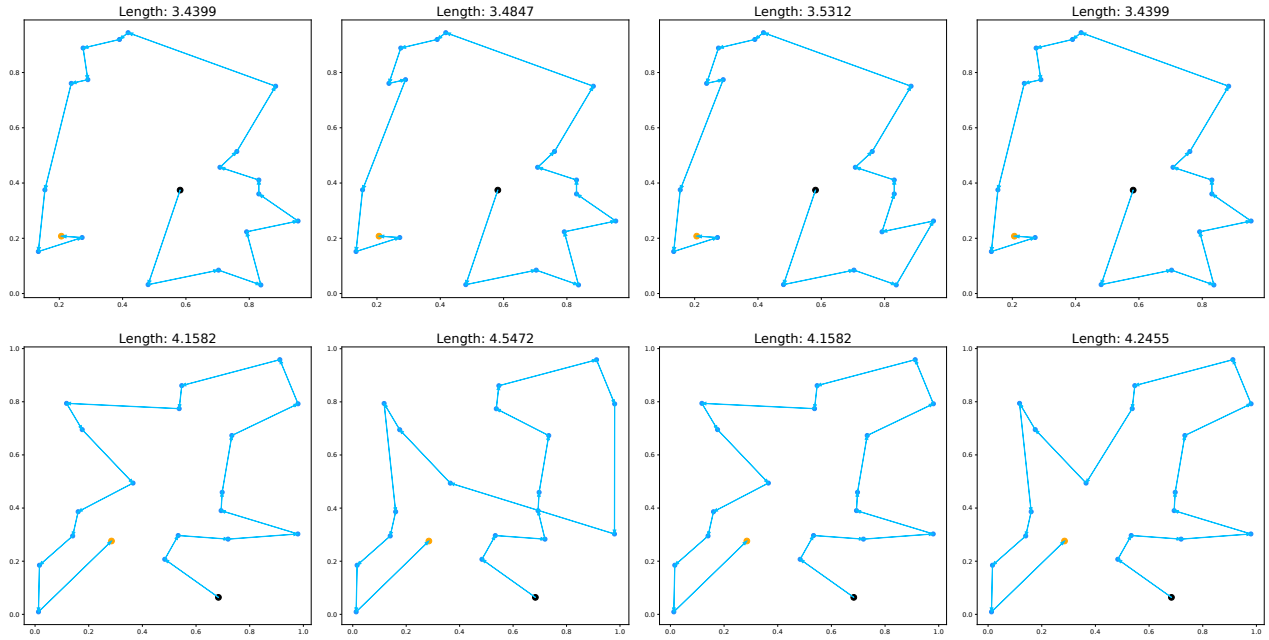
Attention Model *Attention Model* (Kool et al., 2019) is an RL-based autoregressive model for standard TSP. We modify its decoding process so that it can solve TSP-SE and TSP-PRI. Because TSP-SE ends in the ending node instead of constructing a circle, we use the ending node embedding to replace the first node embedding in the original paper during decoding. For TSP-PRI, if the priority node has not been visited within the first $m-1$ steps, it will be visited in the m-th step. The training process and hyper-parameters are the same as our model in Sec. E.1.

MIP, Insertion Heuristic, Standard Solver run on a Intel(R) Core(TM) i7-7820X CPU; *Nearest Neighbor, Attention Model* run on a NVIDIA GeForce RTX 2080Ti GPU with 11GB memory.

E.3. Further Ablation Study

In the following study, we show that LinSAT can normalize matrices outside a neural network. The study involves two variants of our TSP solver presented in Section 4: **1) Random Pre-Projected Matrix**: Apply our LinSAT to a randomly generated matrix and do beam search over it; **2) Trainable Pre-Projected Matrix**: Randomly initialize the pre-projected matrix, view the matrix as trainable parameters and use the same training process as in the main paper to optimize it. **Transformer Feature Matrix** is our TSP solver in Section 4.

The average tour length and total inference time are shown in Table 7. The Random Pre-Projected Matrix cannot provide useful guidance to beam search, thus its performance is poor. Trainable Pre-Projected Matrix performs better, but it is easy to stick at local optima without global features extracted by the neural network. Moreover, updating the pre-projected matrix requires multiple forward and backward passes of LinSAT, making this method much more time-consuming.



(a) Optimal (b) Gurobi (Sec.4.2, 10s) (c) Attention Model (d) LinSAT

Figure 2. Route plan cases of TSP-SE. Each row represents one instance and each column represents one method. Black points stand for starting cities and orange points stand for ending cities. The optimal tours are solved by Gruobi (MTZ).

Table 8. Traveling salesman problem with extra constraints' time cost. Similar to other neural solvers for standard TSP (Kool et al., 2019; Kwon et al., 2021) the post-processing step (e.g. beam search, Monte Carlo tree search) is the most time-consuming, yet it is necessary to achieve better results.

Task	Neural Network	LinSAT	Beam Search
TSP-SE	0.3%	12.4%	87.3%
TSP-PRI	0.3%	12.6%	87.1%

Table 9. Partial graph matching's time cost. Hungarian Top-ϕ is our discretization step discussed at the end of Section 5.3.

Module	Neural Network	LinSAT	Hungarian Top-ϕ
Proportion of Time	17.1%	81.4%	1.5%

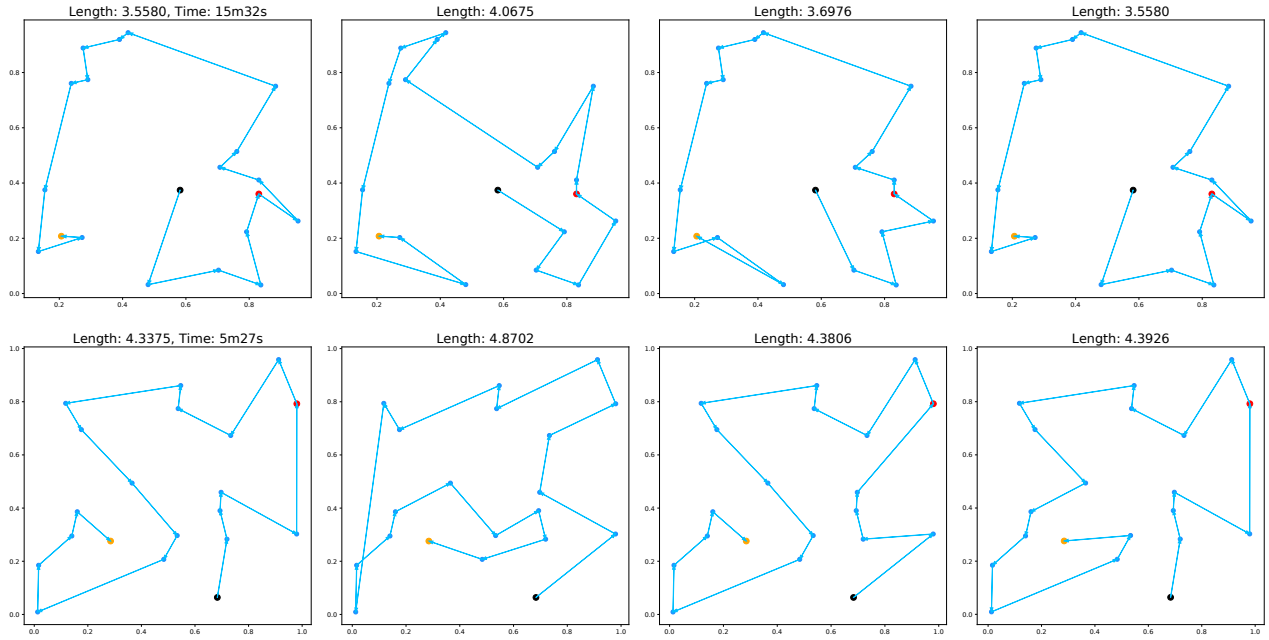
This ablation study proves the feasibility of LinSAT working outside a neural network, while it also shows the necessity of neural networks to get high-quality solutions in an efficient time.

E.4. Visualizations

Fig. 2 and Fig. 3 show some route plan cases of TSP-SE and TSP-PRI using different methods. Our LinSAT is able to get near-optimal solutions in a short time, especially for TSP-PRI where getting an optimal solution is very time-consuming.

F. Experiment Testbed

All our experiments are run on our workstation with Intel(R) Core(TM) i7-7820X CPU, NVIDIA GeForce RTX 2080Ti GPU, and 11GB memory.



(a) Optimal (b) Gurobi (Sec.4.2, 10s) (c) Attention Model (d) LinSAT

Figure 3. Route plan cases of TSP-PRI. Each row represents one instance and each column represents one method. Black points stand for starting cities, orange points stand for ending cities and red points stand for priority cities that need to be visited within the first 5 steps. The optimal tours are solved by Gurobi (Sec.4.2) without a time limit, and the running time to get the corresponding optimal solutions are listed in the titles.

Table 10. Portfolio allocation's time cost. Note that since the decision variables of portfolio allocation are continuous, it does not need any post-processing/discretization steps.

Module	Neural Network	LinSAT
Proportion of Time	20.1%	79.9%

Table 11. Timing statistics of projecting a random matrix in to doubly-stochastic (on CPU, in seconds).

Method	Forward	Backward	Total
LinSAT	0.0382	0.0250	0.0632
CVXPY	0.1344	0.0042	0.1386

G. Study of Time Costs

In our case studies, LinSAT has higher time costs than neural networks, which is in our expectation because the cost of one Sinkhorn iteration could be roughly viewed as one layer of neural network from the unfolding perspective. The complexity of neural networks in our case studies is relatively low, since too many layers may cause the over-smoothing issue, and designing new networks is beyond the scope of this paper. We summarize the proportion of inference time in all 3 case studies in Tables 8 to 10.

Besides, we compare the timing statistics of LinSAT with another regularized projection method – the differentiable CVXPY layers (Agrawal et al., 2019). Specifically, we conduct a case study of transforming a random matrix into a doubly-stochastic matrix. Both methods achieve doubly-stochastic matrices, and the timing statistics are in Table 11. LinSAT is more efficient in this case study. Extra speed-up may be achieved when the input scales up and switching LinSAT to GPU (CVXPY is CPU-only).