FAST SINKHORN II: COLLINEAR TRIANGULAR MATRIX AND LINEAR TIME ACCURATE COMPUTATION OF OPTIMAL TRANSPORT*

QICHEN LIAO†, ZIHAO WANG‡, JING CHEN§, BO BAI¶, SHI JIN‖, AND HAO WU#

Abstract. In our previous work [arXiv:2202.10042], the complexity of Sinkhorn iteration is reduced from $O(N^2)$ to the optimal $O(N)$ by leveraging the special structure of the kernel matrix. In this paper, we explore the special structure of kernel matrices by defining and utilizing the properties of the **L**ower-**Col**-**L**inear **T**riangular Matrix (L-CoLT matrix) and **U**pper-**Col**-**L**inear **T**riangular Matrix (U-CoLT matrix). We prove that (1) L/U-CoLT matrix-vector multiplications can be carried out in $O(N)$ operations; (2) both families of matrices are closed under the Hadamard product and matrix scaling. These properties help to alleviate two key difficulties for reducing the complexity of the Inexact Proximal point method (IPOT), and allow us to significantly reduce the number of iterations to $O(N)$. This yields the Fast Sinkhorn II (FS-2) algorithm for accurate computation of optimal transport with low algorithm complexity and fast convergence. Numerical experiments are presented to show the effectiveness and efficiency of our approach.

Key words. Optimal Transport, Wasserstein-1 metric, Sinkhorn algorithm, IPOT method, FS-2 algorithm

MSC codes. 49M25; 65K10

1. Introduction.

The Wasserstein metric, broadly used in optimal transport theory with applications in many fields including machine learning, quantifies the dissimilarity between two probabilistic distributions. Many methods have been proposed to compute the Wasserstein metrics directly, such as the linear programming methods [30, 22, 36], combinatorial methods [33], solving the Monge-Ampère equations [15, 14, 3], via Benamou-Brenier formulation [2, 21] and the proximal splitting methods [8, 28]. In recent years, several approximation techniques in optimal transport for high-dimensional distributions have also been proposed [26, 27].

The Sinkhorn algorithm [10, 34] is a popular $O(N^2)$ algorithm to approximate the Wasserstein metric [31] by minimizing the entropy regularized optimal transport (OT) problem. In [24], the FS-1 algorithm is proposed to solve entropy regularized OT in $O(N)$ time by leveraging the special structure of the Sinkhorn kernel matrix of the Wasserstein-1 metric. The solution of entropy regularized OT approximates the accurate OT solution only if the regularization parameter is sufficiently small. However, small regularization parameters lead to numerical instability and excessive

*Submitted to the editors DATE.

Funding: This work was supported by National Natural Science Foundation of People's Republic of China Grant Nos.11871297 and 12031013, Tsinghua University Initiative Scientific Research Program, and Shanghai Municipal Science and Technology Major Project 2021SHZDZX0102.

†Department of Mathematical Sciences, Tsinghua University, Beijing 100084, People's Republic of China; Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co. Ltd., Hong Kong SAR, People's Republic of China (lqc20@mails.tsinghua.edu.cn).

‡Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong SAR, People's Republic of China (zwanggc@cse.ust.hk).

§School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 639798 (jing.chen@ntu.edu.sg).

¶Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co. Ltd., Hong Kong SAR, People's Republic of China (baibo8@huawei.com).

‖School of Mathematical Sciences, Institute of Natural Sciences, and MOE-LSC Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China (shijin-m@sjtu.edu.cn).

#Corresponding author. Department of Mathematical Sciences, Tsinghua University, Beijing 100084, People's Republic of China (hwu@tsinghua.edu.cn).

iterations [13]. This causes the slow convergence of the Sinkhorn algorithm.

The Inexact Proximal point method [35] for the Optimal Transport problem (IPOT) has been proposed to address this challenge. It regularizes the original OT by introducing the proximal point term and solves a series of successive subproblems. Only fairly mild regularization parameters are required to ensure the method's fast convergence to the accurate OT solution in an $O(N^2)$ algorithm. The goal of this paper is to construct a new method to accurately compute OT solutions with good convergence behavior and $O(N)$ algorithm complexity by combing the IPOT method and the FS-1 algorithm. Note the two key steps in the IPOT method make it hard to reduce the complexity to $O(N)$: the matrix Hadamard product (Algorithm 2.1, line 4) and the matrix scaling (Algorithm 2.1, line 8). For general matrices, the complexity of the above operations is both $O(N^2)$. Moreover, these operations may destroy the special structure of the kernel matrix [24], making it impossible for us to implement matrix-vector multiplication with $O(N)$ cost.

We will explore the special structure of kernel matrices by defining and exploiting the properties of the **L**ower-**Col**inear **T**riangular Matrix (L-CoLT matrix) and **U**pper-**Col**inear **T**riangular Matrix (U-CoLT matrix). For these matrices, we can realize the matrix-vector multiplication with $O(N)$ cost by using the idea of dynamic programming similar to [24]. Next, we show that each L/U-CoLT matrix can be represented by two vectors of dimension N. Furthermore, we prove the closure of families of L/U-CoLT matrices to matrix Hadamard product and matrix scaling. This means that the special structure of the kernel matrix is preserved by matrix Hadamard product and matrix scaling, so we can still implement matrix-vector multiplication (Algorithm 2.1, lines 6-7) with $O(N)$ cost. On the other hand, by updating two representation vectors of the L/U-CoLT matrix, we can also implement matrix Hadamard product (Algorithm 2.1, line 4) and matrix scaling (Algorithm 2.1, line 8) with $O(N)$ cost. Consequently, the Fast Sinkhorn II (FS-2) algorithm is developed, which integrates the advantages of both IPOT and FS-1. Moreover, we also find that the FS-2 algorithm has the advantage in reducing the space complexity since all the matrices can be represented by vectors. Due to these benefits, one can expect that our FS-2 could be applied in various fields, e.g., machine learning [26, 27, 16, 25], image processing [32, 29], inverse problems [6, 12, 37, 18], density function theory[19, 5, 9].

The rest of the paper is organized as follows. In section 2 , the basics of the Wasserstein-1 metric and the IPOT method are briefly reviewed. After presenting the definition, properties, and fast matrix-vector multiplications of the L/U-CoLT matrix in section 3, we apply them to accelerate the IPOT method, thus developing the FS-2 algorithm in section 4. In section 5, the FS-2 algorithm is extended to high dimensions. The numerical experiments are performed to verify our conclusions in section 6. We conclude the paper in section 7.

2. The Wasserstein-1 metric and the IPOT method. Given two unit discrete distributions $\boldsymbol{u}$ and $\boldsymbol{v}$,

$$\boldsymbol{u} = (u_1, u_2, \cdots u_N)^\top \in \mathbb{R}^N, \quad \boldsymbol{v} = (v_1, v_2, \cdots, v_N)^\top \in \mathbb{R}^N,$$

where $u_i \geq 0$, $v_j \geq 0$, and $\sum_i u_i = \sum_j v_j = 1$. The Wasserstein-1 distance between them is defined as [31]

$$W_1(\boldsymbol{u}, \boldsymbol{v}) = \min_{\Gamma \mathbf{1} = \boldsymbol{u}, \Gamma^T \mathbf{1} = \boldsymbol{v}} \langle C, \Gamma \rangle, \tag{2.1}$$

where $C = [c_{ij}] \in \mathbb{R}^{N\times N}$ is the cost matrix. The element $c_{ij} = \|\boldsymbol{x}_i - \boldsymbol{y}_j\|_1$ represents the cost of transporting the unit mass from position $\boldsymbol{x}_i$ to position $\boldsymbol{y}_j$ and the variable $\Gamma = [\gamma_{ij}] \in \mathbb{R}^{N\times N}$ to be optimized is the transport plan. Here, the Frobenius inner product $\langle A, B\rangle = \sum_{i,j} a_{ij}b_{ij}$, where $A = [a_{ij}], B = [b_{ij}]$ are real-valued matrices.

The Sinkhorn algorithm [10, 34] solves an entropy regularized OT problem to obtain an approximate result of (2.1). However, the small regular parameter required by good approximation leads to a slow convergence rate and numerical instability. To avoid this problem, the proximal point iteration (2.2) is developed to solve (2.1) accurately [35]. It begins with a transport map $\Gamma^{(0)}$ and iteratively solves the following minimization problem

$$\Gamma^{(t+1)} = \underset{\Gamma\mathbf{1}=\boldsymbol{u},\Gamma^T\mathbf{1}=\boldsymbol{v}}{\arg\min} \langle C,\ \Gamma\rangle + \delta^{(t)} D_h\left(\Gamma,\ \Gamma^{(t)}\right), \tag{2.2}$$

where D_h is Bregman divergence, taken in the form of the KL divergence in [35],

$$D_h\left(A,\ B\right) = \sum_{i,j}\left(a_{ij}\ln\frac{a_{ij}}{b_{ij}} - a_{ij} + b_{ij}\right)$$

and $\delta^{(t)}$ is the regular parameter. The Lagrangian of the above equation writes

$$L(\Gamma, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \langle C,\ \Gamma\rangle + \delta^{(t)} D_h\left(\Gamma,\ \Gamma^{(t)}\right) + \boldsymbol{\alpha}^T\left(\Gamma\mathbf{1} - \boldsymbol{u}\right) + \boldsymbol{\beta}^T\left(\Gamma^T\mathbf{1} - \boldsymbol{v}\right).$$

Taking derivative of the Lagrangian with respect of γ_{ij} directly leads to

$$\gamma_{ij} = e^{-\alpha_i/\delta^{(t)}} Q_{ij}^{(t)} e^{-\beta_j/\delta^{(t)}},\ \text{ where }\ Q_{ij}^{(t)} = \gamma_{ij}^{(t)} e^{-c_{ij}/\delta^{(t)}} > 0.$$

Denoting $\odot$ as the Hadamard product, $Q^{(t)} = K \odot \Gamma^{(t)}$ and $K = [e^{-c_{ij}/\delta^{(t)}}] \in \mathbb{R}^{N\times N}$ is the kernel matrix. Letting $\phi_i = e^{-\alpha_i/\delta^{(t)}}$, $\psi_j = e^{-\beta_j/\delta^{(t)}}$, and vectors $\boldsymbol{\phi} = (\phi_i)$ and $\boldsymbol{\psi} = (\psi_j)$, one obtains

$$\mathrm{diag}(\boldsymbol{\phi})Q^{(t)}\mathrm{diag}(\boldsymbol{\psi})\mathbf{1} = \boldsymbol{u}, \quad \mathrm{diag}(\boldsymbol{\psi})Q^{(t)\top}\mathrm{diag}(\boldsymbol{\phi})\mathbf{1} = \boldsymbol{v}. \tag{2.3}$$

By iteratively updating vectors $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$

$$\boldsymbol{\psi}^{(t,\ell+1)} = \boldsymbol{v} \oslash (Q^{(t)\top}\boldsymbol{\phi}^{(t,\ell)}), \quad \boldsymbol{\phi}^{(t,\ell+1)} = \boldsymbol{u} \oslash (Q^{(t)}\boldsymbol{\psi}^{(t,\ell+1)}), \tag{2.4}$$

one can obtain an accurate solution for the original OT problem (2.1). Here $\oslash$ represents pointwise division, t is the proximal iteration step (outer iteration) and ℓ it the Sinkhorn-type iteration step (inner iteration). The pseudo-code of IPOT is shown in Algorithm 2.1.

3. The Collinear Triangular Matrix.

3.1. Definition and Fast Matrix-Vector Multiplication.

DEFINITION 3.1 (Lower/Upper-Collinear Triangular Matrix). *A lower triangular matrix is called a* **Lower-Collinear Triangular Matrix***(L-CoLT matrix) if its corresponding entries on any two rows (columns) have the same–column (row) independent– ratio except those dividing by* 0. *Specifically, the N-dimensional L-CoLT matrix set is defined as follows:*

$$\mathcal{C}_L^N = \left\{ M \in \mathbb{R}^{N\times N} \mid m_{i+1,j}/m_{i,j} = r_i,\ j \le i;\ m_{i,j} = 0,\ i < j,\ \boldsymbol{r} \in (\mathbb{R}\backslash\{0\})^{N-1} \right\}. \tag{3.1}$$

Algorithm 2.1 IPOT

Input: $\boldsymbol{u},\ \boldsymbol{v} \in \mathbb{R}^N$; $K = e^{-C/\delta} \in \mathbb{R}^{N\times N}$; $L, \mathrm{itr\_max} \in \mathbb{N}^+$

Output: $W_1(\boldsymbol{u}, \boldsymbol{v})$

$\boldsymbol{\phi},\ \boldsymbol{\psi} \leftarrow \frac{1}{N}\mathbf{1}_N$
$\Gamma = \mathbf{1}_N\mathbf{1}_N^T$
for $t = 1\ :\ \mathrm{itr\_max}$ **do**
  $Q \leftarrow K \odot \Gamma$
  for $\ell = 1\ :\ L$ **do**
    $\boldsymbol{\psi} \leftarrow \boldsymbol{v} \oslash \left(Q^T\boldsymbol{\phi}\right)$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{u} \oslash \left(Q\boldsymbol{\psi}\right)$
  $\Gamma \leftarrow \mathrm{diag}\left(\boldsymbol{\phi}\right) Q \mathrm{diag}\left(\boldsymbol{\psi}\right)$
return $W_1(\boldsymbol{u}, \boldsymbol{v})$

Similarly, we define **Upper-Collinear Triangular Matrix** *(U-CoLT matrix), which is a strictly upper triangular matrix:*

(3.2)
$$\mathcal{C}_U^N = \left\{ M \in \mathbb{R}^{N\times N} \mid m_{i-1,j}/m_{i,j} = r'_{i-1},\ i < j;\ m_{i,j} = 0,\ i \geq j,\ \boldsymbol{r}' \in \left(\mathbb{R}\backslash\{0\}\right)^{N-2} \right\}.$$

We call the vectors r and r' in (3.1)-(3.2) *the* ***ratio vectors*** *of the collinear triangular matrix.*

The matrices introduced in Definition 3.1 are termed as collinear triangular matrices (CoLT), due to the following collinearity between columns:

$$m_{i,j}/m_{i,j+1} = m_{k,j}/m_{k,j+1} \iff m_{i,j}/m_{k,j} = m_{i,j+1}/m_{k,j+1}.$$

THEOREM 3.2 (Vector Representation of Collinear Triangular Matrix). *Any L-CoLT matrix M_L can be represented by its diagonal elements $\boldsymbol{\gamma}$ and the ratio vector $\boldsymbol{r}$ in Equation* (3.1)*. Any U-CoLT matrix M_U can be represented by its superdiagonal elements $\boldsymbol{\gamma}'$ and the ratio vector $\boldsymbol{r}'$ in* (3.2)*.*

Proof. For any L-CoLT matrix $M_L \in \mathcal{C}_L^N$, if its corresponding $\boldsymbol{\gamma}$ and $\boldsymbol{r}$ are given, then $m_{i,j} = \gamma_j \prod_{k=i}^{N-1} r_k$. The proof of U-CoLT is similar. ∎

In the following, we use L-CoLT$(\boldsymbol{\gamma},\ \boldsymbol{r})$, $\boldsymbol{\gamma} \in \mathbb{R}^N$, $\boldsymbol{r} \in \mathbb{R}^{N-1}$ and U-CoLT$(\boldsymbol{\gamma}',\ \boldsymbol{r}')$, $\boldsymbol{\gamma}' \in \mathbb{R}^{N-1}$, $\boldsymbol{r}' \in \mathbb{R}^{N-2}$ to represent a L-CoLT matrix and a U-CoLT matrix, respectively. A specific correspondence of L-CoLT and U-CoLT is shown as follow:

For the L-CoLT matrix M_L

$$M_L = \text{L-CoLT}(\boldsymbol{\gamma},\ \boldsymbol{r}) = \begin{pmatrix} \gamma_1 & & & & \\ \gamma_1 r_1 & \gamma_2 & & & \\ \gamma_1 r_1 r_2 & \gamma_2 r_2 & \gamma_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \gamma_1 \prod_{i=1}^{N-1} r_i & \gamma_2 \prod_{i=2}^{N-1} r_i & \gamma_3 \prod_{i=3}^{N-1} r_i & \cdots & \gamma_N \end{pmatrix}$$

Similarly, for the U-CoLT matrix M_U

$$M_U = \text{U-CoLT}(\boldsymbol{\gamma}',\ \boldsymbol{r}') = \begin{pmatrix} 0 & \gamma_1' & \gamma_2'r_1' & \cdots & \gamma_{N-1}'\prod_{i=1}^{N-2} r_i' \\ & 0 & \gamma_2' & \cdots & \gamma_{N-1}'\prod_{i=2}^{N-2} r_i' \\ & & \ddots & \ddots & \vdots \\ & & & 0 & \gamma_{N-1}' \\ & & & & 0 \end{pmatrix}$$

The special nature of the L-CoLT and U-CoLT matrices allows us to compute matrix-vector multiplications in $O(N)$ operations.

For any $M_L = \text{L-CoLT}(\boldsymbol{\gamma},\ \boldsymbol{r})$ and vector $\boldsymbol{y} \in \mathbb{R}^N$, the matrix-vector multiplication $M_L\boldsymbol{y}$ is written as

$$M_L\boldsymbol{y} = \begin{pmatrix} \gamma_1 y_1 & + & 0 & + & 0 & \cdots & + & 0 \\ \gamma_1 r_1 y_1 & + & \gamma_2 y_2 & + & 0 & \cdots & + & 0 \\ \gamma_1 r_1 r_2 y_1 & + & \gamma_2 r_2 y_2 & + & \gamma_3 y_3 & \cdots & + & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma_1 \prod_{i=1}^{N-1} r_i y_1 & + & \gamma_2 \prod_{i=2}^{N-1} r_i y_2 & + & \gamma_3 \prod_{i=3}^{N-1} r_i y_3 & \cdots & + & \gamma_N y_N \end{pmatrix}. \tag{3.3}$$

Denote $\boldsymbol{p}_k$ as the summation of the k-th row in (3.3), then one has

$$p_1 = \gamma_1 y_1, \quad p_k = r_{k-1}p_{k-1} + \gamma_k y_k, \quad k = 2, \cdots, N.$$

Based on this recursion formula, a fast implementation is proposed in Algorithm 3.1.

Algorithm 3.1 Fast L-CoLT Matrix-Vector multiplication

Input: input vector $\boldsymbol{y}$ of size N, input matrix $M_L = \text{L-CoLT}(\boldsymbol{\gamma},\ \boldsymbol{r})$
Output: $\boldsymbol{p} = M_L\boldsymbol{y}$

```
procedure LCMV(y, γ, r)
    p_1 = γ_1 y_1
    for i = 1 : N − 1 do
        p_{i+1} = r_i p_i + γ_{i+1} y_{i+1}
return p
```

Similarly, the fast matrix-vector multiplication for U-CoLT matrices is shown in Algorithm 3.2.

Next, we denote the set $\mathcal{C}^N$ as the direct sum of $\mathcal{C}_L^N$ and $\mathcal{C}_U^N$, defined by

DEFINITION 3.3.

$$\mathcal{C}^N = \mathcal{C}_L^N + \mathcal{C}_U^N = \left\{ A + B \mid A \in \mathcal{C}_L^N,\ B \in \mathcal{C}_U^N \right\}. \tag{3.4}$$

Due to the linearity of matrix-vector multiplication, we can further develop the fast matrix-vector multiplication algorithm for matrices in $\mathcal{C}^N$, which is given in Algorithm 3.3.

The space and time complexities of these algorithms are $O(N)$, which is much better than the original matrix-vector multiplication.

Algorithm 3.2 Fast U-CoLT Matrix-Vector multiplication

Input: input vector $\boldsymbol{y}$ of size N, input matrix $M = \text{U-CoLT}(\boldsymbol{\gamma}',\ \boldsymbol{r}')$
Output: $\boldsymbol{q} = M_U\boldsymbol{y}$

procedure UCMV($\boldsymbol{y},\ \boldsymbol{\gamma}',\ \boldsymbol{r}'$)
  $q_N = 0,\ q_{N-1} = \gamma'_{N-1}y_N$
  for $i = 2\ :\ N-1$ **do**
    $q_{N-i} = r'_{N-i}q_{N-i+1} + \gamma'_{N-i}y_{N-i+1}$
return $\boldsymbol{q}$

Algorithm 3.3 CoLT Matrix-Vector multiplication

Input: input vector $\boldsymbol{x}$ of size N, diagonal elements $\boldsymbol{\gamma},\ \boldsymbol{\gamma}'$ and the ratio vector $\boldsymbol{r},\ \boldsymbol{r}'$
Output: $\boldsymbol{p} + \boldsymbol{q} = M\boldsymbol{y}$

procedure CMV($\boldsymbol{y},\ \mathbf{c}^L,\ \mathbf{c}^U,\ \boldsymbol{\gamma},\ \boldsymbol{\gamma}'$)
  $\boldsymbol{p} = \text{LCMV}(\boldsymbol{y},\ \boldsymbol{\gamma},\ \boldsymbol{r})$
  $\boldsymbol{q} = \text{UCMV}(\boldsymbol{y},\ \boldsymbol{\gamma}',\ \boldsymbol{r}')$
return $\boldsymbol{p} + \boldsymbol{q}$

3.2. Some Basic Properties. In this subsection, we justify some basic properties of those matrices involved, which will be used in our algorithm.

THEOREM 3.4. *$\left(\mathcal{C}_L^N,\ \odot\right)$ and $\left(\mathcal{C}_U^N,\ \odot\right)$ are Abelian groups, where $\odot$ is the Hadamard product.*

Proof. We only prove the theorem for $\left(\mathcal{C}_L^N,\ \odot\right)$. It suffices to show that $\left(\mathcal{C}_L^N,\ \odot\right)$ has the following properties:

Closure: For any two matrices $A = \text{L-CoLT}(\hat{\boldsymbol{\gamma}},\ \hat{\boldsymbol{r}})$ and $B = \text{L-CoLT}(\tilde{\boldsymbol{\gamma}},\ \tilde{\boldsymbol{r}})$, we set $D = A \odot B$. Since $d_{i,j} = a_{i,j}b_{i,j}$, one has

$$d_{i,j}/d_{i+1,j} = (a_{i,j}b_{i,j})\,/\,(a_{i+1,j}b_{i+1,j}) = \hat{r}_i\tilde{r}_i,\ j = 1, 2, \cdots, i, \tag{3.5}$$

and the strictly upper triangle part of D is obviously 0, which means $D = \text{L-CoLT}(\hat{\boldsymbol{\gamma}} \odot \tilde{\boldsymbol{\gamma}},\ \hat{r} \odot \tilde{r}) \in \mathcal{C}_L^N$.

Identity and Inverses: Let

$$E = \begin{pmatrix} 1 & & & & \\ 1 & 1 & & & \\ 1 & 1 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \in \mathcal{C}_L^N, \tag{3.6}$$

then for any $A = \text{L-CoLT}(\boldsymbol{\gamma},\ \boldsymbol{r})$, $A \odot E = E \odot A = A$, which means E is the identity element. Let

$$B = \begin{pmatrix} 1/a_{11} & & & & \\ 1/a_{21} & 1/a_{22} & & & \\ 1/a_{31} & 1/a_{32} & 1/a_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1/a_{n1} & 1/a_{n2} & 1/a_{n3} & \cdots & 1/a_{nn} \end{pmatrix}.$$

Since

$$b_{i,j}/b_{i+1,j} = a_{i+1,j}/a_{i,j} = 1/r_i,\ j = 1, 2, \cdots, i,$$

∎

then $B \in \mathcal{C}_L^N$. And obviously, $A \odot B = B \odot A = E$, which means B is the inverse of A.

Commutativity and Associativity: The commutativity and associativity can be derived from the commutative and associative law of real number multiplication.

Based on the above theorem, we can deduce directly

COROLLARY 3.5. *$\left(\mathcal{C}^N,\ \odot\right)$ is an abelian group with identity element $\mathbf{1}_N \mathbf{1}_N^T$.*

THEOREM 3.6. *For any vector $\boldsymbol{x} \in \left(\mathbb{R}\backslash\{0\}\right)^N$, $f_{\boldsymbol{x}}(M) = (diag(\boldsymbol{x}))\, M$ and $g_{\boldsymbol{x}}(M) = M\,(diag(\boldsymbol{x}))$ are permutations in $\mathcal{C}_L^N$ and $\mathcal{C}_U^N$.*

Proof. We only prove the theorem for $\mathcal{C}_L^N$.

Closure: For any vector $\boldsymbol{x} \in \left(\mathbb{R}\backslash\{0\}\right)^N$, and $M_L = \text{L-CoLT}(\boldsymbol{\gamma},\ \boldsymbol{r})$, let E be the one defined in Equation (3.6). Since

$$E_1 = (\text{diag}\,(\boldsymbol{x}))\, E = \begin{pmatrix} x_1 & & & & \\ x_2 & x_2 & & & \\ x_3 & x_3 & x_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ x_n & x_n & x_n & \cdots & x_n \end{pmatrix} \in \mathcal{C}_L^N,$$

$$E_2 = E\,(\text{diag}\,(\boldsymbol{x})) = \begin{pmatrix} x_1 & & & & \\ x_1 & x_2 & & & \\ x_1 & x_2 & x_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ x_1 & x_2 & x_3 & \cdots & x_n \end{pmatrix} \in \mathcal{C}_L^N,$$

we have

$$\begin{aligned} &f_{\boldsymbol{x}}(M) = (\text{diag}\,(\boldsymbol{x}))\, M = (\text{diag}\,(\boldsymbol{x}))\, E \odot M = E_1 \odot M \in \mathcal{C}_L^N; \\ &g_{\boldsymbol{x}}(M) = M\,(\text{diag}\,(\boldsymbol{x})) = M \odot E\,(\text{diag}\,(\boldsymbol{x})) = M \odot E_2 \in \mathcal{C}_L^N. \end{aligned} \tag{3.7}$$

The last set membership can be derived by the closure of $\left(\mathcal{C}_L^N,\ \odot\right)$ proved in Theorem 3.4. Hence, $f_{\boldsymbol{x}}$ and $g_{\boldsymbol{x}}$ are maps from $\mathcal{C}_L^N$ to itself.

Injectiveness: For any two matrices $A = \text{L-CoLT}(\hat{\boldsymbol{\gamma}},\ \hat{r})$ and $B = \text{L-CoLT}(\tilde{\boldsymbol{\gamma}},\ \tilde{\boldsymbol{r}})$, let D_1 be the inverse of E_1 and D_2 be the inverse of E_2. If $f_{\boldsymbol{x}}(A) = f_{\boldsymbol{x}}(B)$, then

$$A = D_1 \odot E_1 \odot A = D_1 \odot f_{\boldsymbol{x}}(A) = D_1 \odot f_{\boldsymbol{x}}(B) = D_1 \odot E_1 \odot B = B.$$

If $g_{\boldsymbol{x}}(A) = g_{\boldsymbol{x}}(B)$, then

$$A = A \odot E_2 \odot D_2 = g_{\boldsymbol{x}}(A) \odot D_2 = g_{\boldsymbol{x}}(B) \odot D_2 = B \odot E_2 \odot D_2 = B,$$

which means $f_{\boldsymbol{x}}(\cdot)$ and $g_{\boldsymbol{x}}(\cdot)$ are injective functions.

Surjectiveness: For any $M \in \mathcal{C}_L^N$, let $Q_1 = D_1 \odot M$ and $Q_2 = M \odot D_2$, then

$$\begin{aligned} &f_{\boldsymbol{x}}(Q_1) = E_1 \odot D_1 \odot M = E \odot M = M; \\ &g_{\boldsymbol{x}}(Q_2) = M \odot D_2 \odot E_2 = M \odot E = M, \end{aligned}$$

which means $f_{\boldsymbol{x}}(\cdot)$ and $g_{\boldsymbol{x}}(\cdot)$ are surjective functions. ∎

COROLLARY 3.7. *$f_{\boldsymbol{x}}(\cdot)$ and $g_{\boldsymbol{x}}(\cdot)$ are permutations in $\mathcal{C}^N$.*

4. The Fast Sinkhorn II. In this section, we will discuss the implementation details to accelerate IPOT. In Algorithm 2.1, three parts lead to $O(N^2)$ algorithm complexity, i.e., the matrix Hadamard product (line 4), the matrix-vector multiplication (lines 6-7), and the matrix scaling (line 8). They all rely on the representation and manipulation of L/U CoLT matrices.

For two discrete distributions on a 1D uniform mesh grid with a grid spacing of h, by introducing the notation $\lambda = e^{-h/\delta}$, the kernel matrix K is written as

$$K = \begin{pmatrix} 1 & \lambda & \lambda^2 & \cdots & \lambda^{N-1} \\ \lambda & 1 & \lambda & \cdots & \lambda^{N-2} \\ \lambda^2 & \lambda & 1 & \cdots & \lambda^{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda^{N-1} & \lambda^{N-2} & \lambda^{N-3} & \cdots & 1 \end{pmatrix} \in \mathcal{C}^N. \tag{4.1}$$

Below we discuss step by step of IPOT (Algorithm 2.1) to reduce the complexity:

- line 2: $\Gamma = \mathbf{1}_N \mathbf{1}_N^T \in \mathcal{C}^N$, we only need four vectors $(\boldsymbol{\gamma}, \boldsymbol{\gamma}', \boldsymbol{r}, \boldsymbol{r}')$ to represent Γ according to Theorem 3.2.
- line 4: the matrix Hadamard product $Q = K \odot \Gamma \in \mathcal{C}^N$ according to Theorem 3.4 and Corollary 3.5. By updating the four representation vectors $(\boldsymbol{\gamma}, \boldsymbol{\gamma}', \boldsymbol{r}, \boldsymbol{r}')$, we can obtain Q with $O(N)$ cost.
- lines 6-7: the matrix-vector multiplication $Q^T\boldsymbol{\phi}$ and $Q\boldsymbol{\psi}$ can be implemented with $O(N)$ cost according to Algorithm 3.3.
- line 8: the matrix scaling $\Gamma = \mathrm{diag}(\boldsymbol{\phi}) Q \mathrm{diag}(\boldsymbol{\psi}) \in \mathcal{C}^N$ according to Theorem 3.6 and Corollary 3.7. By updating the four representation vectors $(\boldsymbol{\gamma}, \boldsymbol{\gamma}', \boldsymbol{r}, \boldsymbol{r}')$, we can obtain Γ with $O(N)$ cost.

Based on the above discussions, we proposed the FS-2 algorithm with $O(N)$ complexity. The pseudo-code is presented in Algorithm 4.1.

Algorithm 4.1 1D FS-2 Algorithm

Input: $\boldsymbol{u},\ \boldsymbol{v} \in \mathbb{R}^N$; $L, \mathrm{itr\_max} \in \mathbb{N}^+$; $h, \delta \in \mathbb{R}$
Output: $W_1(\boldsymbol{u}, \boldsymbol{v})$

1: $\lambda \leftarrow e^{-h/\delta}$; $\boldsymbol{\phi}, \boldsymbol{\psi} \leftarrow \frac{1}{N}\mathbf{1}_N$; $\boldsymbol{r}, \boldsymbol{s} \leftarrow \mathbf{0}_N$
2: $\boldsymbol{\alpha}^L,\ \boldsymbol{\beta}^L,\ \boldsymbol{\alpha}^U,\ \boldsymbol{\beta}^U \leftarrow \lambda \mathbf{1}_{N-1}$; $\boldsymbol{\gamma} \leftarrow \mathbf{1}_N$; $\boldsymbol{\gamma}' \leftarrow \lambda \mathbf{1}_{N-1}$
3: **for** $t = 1 : \mathrm{itr\_max}$ **do**
4: **for** $\ell = 1 : L$ **do**
5: $\boldsymbol{r} \leftarrow \mathrm{CMV}(\boldsymbol{\phi}, \boldsymbol{\beta}^L, \boldsymbol{\beta}^U, \boldsymbol{\gamma}, \boldsymbol{\gamma}')$
6: $\boldsymbol{\psi} \leftarrow \boldsymbol{v} \oslash \boldsymbol{r}$
7: $\boldsymbol{s} \leftarrow \mathrm{CMV}(\boldsymbol{\psi}, \boldsymbol{\alpha}^L, \boldsymbol{\alpha}^U, \boldsymbol{\gamma}, \boldsymbol{\gamma}')$
8: $\boldsymbol{\phi} \leftarrow \boldsymbol{u} \oslash \boldsymbol{s}$
9: **for** $i = 1 : N-1$ **do**
10: $\alpha_i^L \leftarrow \lambda \alpha_i^L\, (\phi_{i+1}/\phi_i)$, $\beta_i^L \leftarrow \lambda \beta_i^L\, (\psi_{i+1}/\psi_i)$
11: $\gamma_i' \leftarrow \lambda \gamma_i' \phi_i \psi_{i+1}$
12: **for** $i = 1 : N-2$ **do**
13: $\alpha_i^U \leftarrow \lambda \alpha_i^U\, (\phi_i/\phi_{i+1})$, $\beta_i^U \leftarrow \lambda \beta_i^U\, (\psi_i/\psi_{i+1})$
14: $\boldsymbol{\gamma} \leftarrow \boldsymbol{\phi} \odot \boldsymbol{\psi} \odot \boldsymbol{\gamma}$
return $W_1(\boldsymbol{u}, \boldsymbol{v})$

There is a minor flaw in the above algorithm. The computational cost of $W_1(\boldsymbol{u}, \boldsymbol{v})$ is still $O(N^2)$ in the last step. This was also ignored in our previous paper [24]. Now,

we would like to discuss this issue. The computation of $W_1(\boldsymbol{u},\boldsymbol{v}) = \langle C,\ \Gamma\rangle$ can be regarded as the summation of all elements of the following matrix.

(4.2)
$$C \odot \Gamma = \begin{pmatrix} 0 & h\gamma_1' & 2h\gamma_2' r_1' & \cdots & (N-1)h\gamma_{N-1}' \prod_{i=1}^{N-2} r_i' \\ h\gamma_1 r_1 & 0 & h\gamma_2' & \cdots & (N-2)h\gamma_{N-1}' \prod_{i=2}^{N-2} r_i' \\ 2h\gamma_1 r_1 r_2 & h\gamma_2 r_2 & 0 & \cdots & (N-3)h\gamma_{N-1}' \prod_{i=3}^{N-2} r_i' \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (N-1)\, h\gamma_1 \prod_{i=1}^{N-1} r_i & (N-2)\, h\gamma_2 \prod_{i=2}^{N-1} r_i & (N-3)\, h\gamma_3 \prod_{i=3}^{N-1} r_i & \cdots & 0 \end{pmatrix}.$$

We separate the summation of the matrix to the lower and strictly upper triangular parts. Thus, the k-th line summation of two parts can be written as

$$p_k = \sum_{i=1}^{k} \omega_{ki}, \quad q_k = \sum_{i=k+1}^{N} \omega_{ki}.$$

We can consider the following recursive computation

$$\begin{aligned} &p_1 = 0, \quad p_2 = h\gamma_1 r_1, \quad p_2' = h\gamma_1 r_1 + h\gamma_2, \\ &p_i = r_{i-1}\left(p_{i-1} + p_{i-1}'\right), \quad p_i' = r_{i-1} p_{i-1}' + h\gamma_i, \quad i = 3,\ 4, \cdots,\ N. \\ &q_N = 0, \quad q_{N-1} = h\gamma_{N-1}', \quad q_{N-1}' = h\gamma_{N-1}' + h, \\ &q_j = r_j'\left(q_{j+1} + q_{j+1}'\right), \quad q_j' = r_j' q_{j+1}' + h, \quad j = 1,\ 2, \cdots,\ N-2. \end{aligned} \tag{4.3}$$

Thus, the Wasserstein-1 metric can be finally obtained with $O(N)$ cost

$$W_1(\boldsymbol{u},\boldsymbol{v}) = \langle C,\ \Gamma\rangle = \sum_{i=1}^{N} \left(p_i + q_i\right).$$

5. Extension to high dimension. In this section, we illustrate how the FS-2 algorithm generalizes to higher dimensions using the two-dimensional case as an example.

5.1. Block Collinear Triangular Matrix. Hereinafter, for $A \in \mathbb{R}^{MN\times MN}$, we break it into M^2 uniform blocks with size $N \times N$:

$$A = \left(\begin{array}{c|c|c|c|c} A_{1,1} & A_{1,2} & A_{1,3} & \cdots & A_{1,M} \\ \hline A_{2,1} & A_{2,2} & A_{2,3} & \cdots & A_{2,M} \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline A_{M,1} & A_{M,2} & A_{M,3} & \cdots & A_{M,M} \end{array}\right).$$

And for vectors $\boldsymbol{x} \in \mathbb{R}^{kN}$, we break it into k uniform blocks as $(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k)^T$, in which

$$\boldsymbol{x}_i = \left(x_{1+(i-1)N},\ x_{2+(i-1)N},\ \cdots,\ x_{iN}\right)^T, \quad i = 1, 2, \cdots, k.$$

To carry out the fast implementation of the matrix-vector multiplication of the block matrix above, we generalize the definition of $\mathcal{C}^N$ in (3.4) to the block case as

DEFINITION 5.1.

$$\mathcal{C}^{N,M} = \{A \in \mathbb{R}^{MN\times MN} | A_{k,k} \in \mathcal{C}^N;\ \boldsymbol{r}^L, \boldsymbol{r}^U \in (\mathbb{R}\backslash\{0\})^{(M-1)N};\\ A_{i+1,j} = \left(diag\left(\boldsymbol{r}_i^L\right)\right) A_{i,j},\ j \le i;\ A_{i-1,j} = \left(diag\left(\boldsymbol{r}_{i-1}^U\right)\right) A_{i,j},\ i \le j\}.$$

Since $\mathcal{C}^{N,M}$ is a generalization of $\mathcal{C}^N$, we can also use the strategy of Algorithm 3.3 in blocks to reduce the computational cost of matrix-vector multiplications. For a vector $\boldsymbol{x} \in \mathbb{R}^{NM}$, the matrix-vector multiplication $A\boldsymbol{x}$ is written as

$$A\boldsymbol{x} = \begin{pmatrix} A_{1,1}\boldsymbol{x}_1 & + & A_{1,2}\boldsymbol{x}_2 & + & A_{1,3}\boldsymbol{x}_3 & \cdots & + & A_{1,M}\boldsymbol{x}_M \\ A_{2,1}\boldsymbol{x}_1 & + & A_{2,2}\boldsymbol{x}_2 & + & A_{2,3}\boldsymbol{x}_3 & \cdots & + & A_{2,M}\boldsymbol{x}_M \\ A_{3,1}\boldsymbol{x}_1 & + & A_{3,2}\boldsymbol{x}_2 & + & A_{3,3}\boldsymbol{x}_3 & \cdots & + & A_{3,M}\boldsymbol{x}_M \\ \vdots & & \vdots & & \vdots & \ddots & \vdots & \vdots \\ A_{M,1}\boldsymbol{x}_1 & + & A_{M,2}\boldsymbol{x}_2 & + & A_{M,3}\boldsymbol{x}_3 & \cdots & + & A_{M,M}\boldsymbol{x}_M \end{pmatrix}. \tag{5.1}$$

We separate the summation of row k to the lower triangular part $\boldsymbol{p}_k$ and the strictly upper triangular part $\boldsymbol{q}_k$. Then computing $A\boldsymbol{x}$ is formulated as

$$A\boldsymbol{x} = \boldsymbol{p} + \boldsymbol{q}, \quad \boldsymbol{p}_k = \sum_{i=1}^{k} A_{k,i}\boldsymbol{x}_i, \quad \boldsymbol{q}_k = \sum_{i=k+1}^{M} A_{k,i}\boldsymbol{x}_i, \quad k = 1, \cdots, M.$$

If A is in $\mathcal{C}^{N,M}$ with $\boldsymbol{r}^L$ and $\boldsymbol{r}^U$, instead of directly calculating $\boldsymbol{p}_k$ and $\boldsymbol{q}_k$, a successive computation is used

$$\begin{aligned} &\boldsymbol{p}_1 = A_{1,1}\boldsymbol{x}_1, \quad \boldsymbol{p}_k = \boldsymbol{r}_{k-1}^L \odot \boldsymbol{p}_{k-1} + A_{k,k}\boldsymbol{x}_k, \quad k = 2, \cdots, M, \\ &\boldsymbol{q}_M = \boldsymbol{0}_N, \quad \boldsymbol{q}_{k-1} = \boldsymbol{r}_{k-1}^U \odot (\boldsymbol{q}_k + A_{k,k}\boldsymbol{x}_k), \quad k = M, M-1, \cdots, 2. \end{aligned} \tag{5.2}$$

Since the computation of $A_{k,k}\boldsymbol{x}_k$ can be carried out with $O(N)$ complexity by using Algorithm 3.3, the whole computation is of $O(NM)$ complexity.

Similar to Theorem 3.4 and Theorem 3.6, $\mathcal{C}^{N,M}$ is closed under the Hadamard product and matrix scaling.

THEOREM 5.2. *$\left(\mathcal{C}^{N,M}, \odot\right)$ is an Abelian group; Matrix scaling operations are permutations in $\mathcal{C}^{N,M}$.*

Proof. For any $A \in \mathcal{C}^{N,M}$, since the diagonal blocks of A are in $\mathcal{C}^N$, by Corollary 3.7, all blocks in A are in $\mathcal{C}^N$. Then the two properties can be proved in a similar way as in subsection 3.2. ∎

5.2. The 2D FS-2 Algorithm. Consider two discretized probabilistic distributions

$$\boldsymbol{u} = (u_{11}, u_{21}, \cdots, u_{N1}, u_{12}, \cdots, u_{i_1 j_1}, \cdots, u_{NM}),$$
$$\boldsymbol{v} = (v_{11}, v_{21}, \cdots, v_{N1}, v_{12}, \cdots, v_{i_2 j_2}, \cdots, v_{NM}),$$

on a uniform 2D mesh of size $N \times M$ with a vertical spacing of h_1 and a horizontal spacing of h_2. The corresponding kernel matrix is written as

$$K = \left(\begin{array}{c|c|c|c|c} K_0 & \lambda_2 K_0 & \lambda_2^2 K_0 & \cdots & \lambda_2^{M-1} K_0 \\ \hline \lambda_2 K_0 & K_0 & \lambda_2 K_0 & \cdots & \lambda_2^{M-2} K_0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \lambda_2^{M-1} K_0 & \lambda_2^{M-2} K_0 & \lambda_2^{M-3} K_0 & \cdots & K_0 \end{array}\right),$$

where the sub-matrix

$$K_0 = \begin{pmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{N-1} \\ \lambda_1 & 1 & \cdots & \lambda_1^{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^{N-1} & \lambda_1^{N-2} & \cdots & 1 \end{pmatrix},$$

and

$$\lambda_1 = e^{-h_1/\delta}, \quad \lambda_2 = e^{-h_2/\delta}.$$

Obviously, the original 2D kernel K contains blocks which are multiples of the 1D kernel, hence belongs to $\mathcal{C}^{N,M}$. By an analysis similar to that in section 4 and using Theorem 5.2, the matrices Q and Γ are in $\mathcal{C}^{N,M}$ throughout the course of the iteration, which means that all the matrix-vector multiplications can be carried out by using recursion (5.2). Thus, the total cost of matrix-vector multiplication of our FS-2 algorithm for 2D Wasserstein-1 metric is reduced to $O(NM)$.

In the 2D FS-2 algorithm, we use '$\widehat{\ }$' to distinguish whether it is a coefficient of the block or the inner sub-matrix, and update them simultaneously after an inner loop. The pseudo-code is presented in Algorithm 5.1. Updating the coefficients of inner sub-matrices and computation of $W_1(\boldsymbol{u}, \boldsymbol{v})$ are omitted in the pseudo-code since they are similar to the 1D case, which we have described in detail.

6. Numerical Experiments. In this section, we carry out three numerical experiments to evaluate the FS-2 algorithm, including one 1D example and two 2D examples. The true Wasserstein metric W_{LP} is obtained by solving the original OT (2.1) using interior-point methods [11, 20]. In our experiments, for both IPOT and FS-2, the number of inner loops is set as $L = 20$ and the regularization parameter $\delta^{(t)}$ is set to 1. The number of iterations here is the total number of loops: #iteration = itr_max $\times L$. In order to deal with the difficulties caused by zeros, we utilize the rescaling method in [23]:

$$D(f,g) = W_1\left(\frac{\frac{f}{\|f\|} + \eta}{1 + N\eta}, \frac{\frac{g}{\|g\|} + \eta}{1 + N\eta}\right), \tag{6.1}$$

In the following, we refer to formula (6.1) for numerical stability with $\eta = 10^{-5}$. All the experiments are conducted on a platform with 128G RAM, and one Intel(R) Xeon(R) Gold 5117 CPU @2.00GHz with 14 cores.

Algorithm 5.1 2D FS-2 Algorithm

Input: $\boldsymbol{u},\ \boldsymbol{v} \in \mathbb{R}^{NM}$; $L, \text{itr\_max} \in \mathbb{N}^+$; $h_1, h_2, \delta \in \mathbb{R}$
Output: $W_1(\boldsymbol{u}, \boldsymbol{v})$

$\lambda_1 \leftarrow e^{-h_1/\delta}$; $\lambda_2 \leftarrow e^{-h_2/\delta}$; $\boldsymbol{\phi}^{(0)}, \boldsymbol{\psi}^{(0)} \leftarrow \frac{1}{NM}\mathbf{1}_{NM}$; $\boldsymbol{p}^{(0)}, \boldsymbol{r}^{(0)}, \boldsymbol{q}^{(0)}, \boldsymbol{s}^{(0)} \leftarrow \mathbf{0}_{NM}$
$\boldsymbol{\gamma} \leftarrow \mathbf{1}_{NM},\ \boldsymbol{\gamma}',\ \boldsymbol{\alpha}^L,\ \boldsymbol{\beta}^L, \lambda_1\mathbf{1}_{(N-1)M},\ \boldsymbol{\alpha}^U,\ \boldsymbol{\beta}^U \leftarrow \lambda_1\mathbf{1}_{(N-2)M}$
$\widehat{\boldsymbol{\alpha}}^L,\ \widehat{\boldsymbol{\beta}}^L,\ \widehat{\boldsymbol{\alpha}}^U,\ \widehat{\boldsymbol{\beta}}^U \leftarrow \lambda\mathbf{1}_{N(M-1)}$
while $t = 1\ :\ \text{itr\_max}$ **do**
  for $\ell = 1\ :\ L$ **do**
    $\boldsymbol{r}_1 \leftarrow \text{CMV}(\boldsymbol{\phi}_1,\ \boldsymbol{\beta}_1)$, $s_M \leftarrow \mathbf{0}_N$
    for $i = 1\ :\ M-1$ **do**
      $\boldsymbol{r}_{i+1} \leftarrow \widehat{\boldsymbol{\beta}}_i^L \odot \boldsymbol{r}_i + \text{CMV}(\boldsymbol{\phi}_{i+1}, \boldsymbol{\beta}_{i+1}^L, \boldsymbol{\beta}_{i+1}^U, \boldsymbol{\gamma}_{i+1}, \boldsymbol{\gamma}'_{i+1})$
      $\boldsymbol{s}_{N-i} \leftarrow \widehat{\boldsymbol{\beta}}_{N-i}^U \odot (\boldsymbol{s}_{N-i+1} + \text{CMV}(\boldsymbol{\phi}_{N-i+1}, \boldsymbol{\beta}_{N-i+1}^L, \boldsymbol{\beta}_{N-i+1}^U, \boldsymbol{\gamma}_{N-i+1}, \boldsymbol{\gamma}'_{N-i+1}))$
    $\boldsymbol{\psi} \leftarrow \boldsymbol{v} \oslash (\boldsymbol{r} + \boldsymbol{s})$
    $\boldsymbol{p}_1 \leftarrow \text{CMV}(\boldsymbol{\psi}_1,\ \boldsymbol{\alpha}_1)$, $\boldsymbol{q}_M \leftarrow \mathbf{0}_N$
    for $i = 1\ :\ M-1$ **do**
      $\boldsymbol{p}_{i+1} \leftarrow \widehat{\boldsymbol{\alpha}}_i^L \odot \boldsymbol{p}_i + \text{CMV}(\boldsymbol{\psi}_{i+1}, \boldsymbol{\alpha}_{i+1}^L, \boldsymbol{\alpha}_{i+1}^U, \boldsymbol{\gamma}_{i+1}, \boldsymbol{\gamma}'_{i+1})$
      $\boldsymbol{q}_{N-i} \leftarrow \widehat{\boldsymbol{\alpha}}_{N-i}^U \odot (\boldsymbol{q}_{N-i+1} + \text{CMV}(\boldsymbol{\psi}_{N-i+1}, \boldsymbol{\alpha}_{N-i+1}^L, \boldsymbol{\alpha}_{N-i+1}^U, \boldsymbol{\gamma}_{N-i+1}, \boldsymbol{\gamma}'_{N-i+1}))$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{u} \oslash (\boldsymbol{p} + \boldsymbol{q})$
  for $i = 1\ :\ M-1$ **do**
    $\widehat{\boldsymbol{\alpha}}_i^L \leftarrow \lambda_2\left(\boldsymbol{\phi}_{i+1}/\boldsymbol{\phi}_i\right) \odot \widehat{\boldsymbol{\alpha}}_i^L$, $\widehat{\boldsymbol{\beta}}_i^L \leftarrow \lambda_2\left(\boldsymbol{\psi}_{i+1}/\boldsymbol{\psi}_i\right) \odot \widehat{\boldsymbol{\beta}}_i^L$
    $\widehat{\boldsymbol{\alpha}}_i^U \leftarrow \lambda_2\left(\boldsymbol{\phi}_i/\boldsymbol{\phi}_{i+1}\right) \odot \widehat{\boldsymbol{\alpha}}_i^U$, $\widehat{\boldsymbol{\beta}}_i^U \leftarrow \lambda_2\left(\boldsymbol{\psi}_i/\boldsymbol{\psi}_{i+1}\right) \odot \widehat{\boldsymbol{\beta}}_i^U$
  Update $\boldsymbol{\gamma},\ \boldsymbol{\gamma}',\ \boldsymbol{\alpha}^L,\ \boldsymbol{\beta}^L,\ \boldsymbol{\alpha}^U,\ \boldsymbol{\beta}^U$
return $W_1(\boldsymbol{u}, \boldsymbol{v})$

6.1. 1D Gaussian distributions. We consider the Wasserstein-1 metric between two mixtures of 1D Gaussian distributions: $0.4\mathcal{N}(60, 64) + 0.6\mathcal{N}(40, 36)$ and $0.5\mathcal{N}(35, 81) + 0.5\mathcal{N}(70, 81)$, which is the experiment setting in [35]. Input vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are generated by integration on the uniform discretization of interval $[0, 100]$ with node size N.

We first compare the convergence of FS-1 and FS-2 for $N = 1000$. We tested 100 experiments, and each experiment was performed for $10,000$ iterations. In Figure 1, the differences of the Wasserstein-1 metric between the true solution W_{LP} and the numerical solutions generated by FS-1 and FS-2 are depicted. As expected, as ε decreases, the error of FS-1 decreases gradually after the iterations converge. We can observe this for $\varepsilon = 1/20$ and $\varepsilon = 1/80$. However, for $\varepsilon = 1/320$, we can not observe convergence. In fact, the error does not drop over $10,000$ iterations. This is because ε is too small, making updates extremely slow. In fact, after $20,000$ iterations, the result of $\varepsilon = 1/320$ will continue to drop, and the final error is smaller than that of $\varepsilon = 1/20$ and $\varepsilon = 1/80$. However, in any case, the results of FS-1 are far inferior to those of FS-2, both in terms of accuracy and convergence rate.

The averaged computational time of the IPOT method and the FS-2 algorithm is given in Table 1 and Figure 2 (left). Apparently, the FS-2 algorithm has achieved an overwhelming advantage over the IPOT method in terms of computational speed, and ensures that the transport plans of the two are almost the same. This replicates the advantages of the FS-1 algorithm over the Sinkhorn algorithm. According to the data

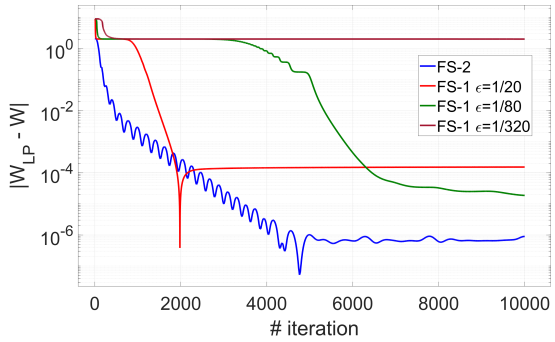

FIG. 1. *The 1D Gaussian distribution problem. The errors between the numerical results generated by FS-1 or FS-2 and the true Wasserstein-1 metric w.r.t. number of iterations.*

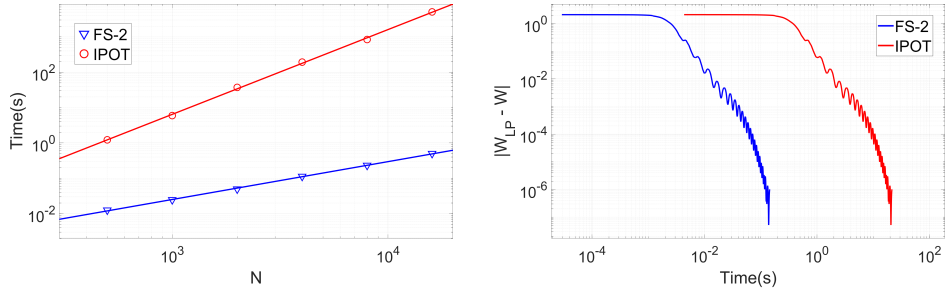

FIG. 2. *The 1D Gaussian distribution problem. Left: The comparison of computational time between the FS-2 algorithm and the IPOT method with different numbers of grid points N. Right: The computational time required to reach the absolute error of the Wasserstein-1 metric.*

fitting results, the empirical complexity of the FS-2 algorithm is $O(N^{1.07})$, which is much smaller than the $O(N^{2.40})$ complexity of the IPOT method. At last, we show the computational time required to reach the absolute error of the Wasserstein-1 metric for $N = 1,000$ in Figure 2 (right). Clearly, the FS-2 algorithm has an advantage of two orders of magnitude in computational time compared to the IPOT method.

6.2. 2D Random distributions. Next, we compute the Wasserstein-1 metric between two $N \times N$ dimensional random vectors whose elements obey the uniform distribution on $(0, 1)$. Without loss of generality, we set $h_1 = h_2 = 0.1$. We also tested 100 experiments, and each experiment was performed for $10,000$ iterations. We hope to test the performance of the FS-2 algorithm in 2D through this example. The differences in the Wasserstein-1 metric between the true solution W_{LP} and the numerical solutions generated by FS-1 and FS-2 are shown in Figure 3. From this, we can observe that FS-1 converges quickly for $\varepsilon = 1/20$, but the error is large. When $\varepsilon = 1/80$, the iteration converges at about $5,000$ steps. The error keep decreasing even after $10,000$ steps for $\varepsilon = 1/320$. However, their errors and convergence speed are not as good as FS-2.

The averaged computational time of the IPOT method and the FS-2 algorithm is given in Table 2 and Figure 4 (left). According to the data fitting results, the empirical complexity of the FS-2 algorithm is $O(N^{2.05})$, which is much smaller than

TABLE 1

The 1D Gaussian distribution problem. The comparison between the IPOT method and the FS-2 algorithm with the different number of grid points N. Columns 2-4 are the averaged computational time of the two algorithms and the speed-up ratio of the FS-2 algorithm. Column 5 is the Frobenius norm of the difference between the transport plan computed by the two algorithms.

N	Computational time (s)		Speed-up ratio	$\|P_{FS} - P\|_F$
	FS-2	IPOT		
500	1.25×10^{-2}	1.22×10^{0}	9.76×10^{1}	2.09×10^{-15}
2000	4.95×10^{-2}	3.73×10^{1}	7.52×10^{2}	6.65×10^{-16}
8000	2.32×10^{-1}	8.41×10^{2}	3.63×10^{3}	8.57×10^{-16}

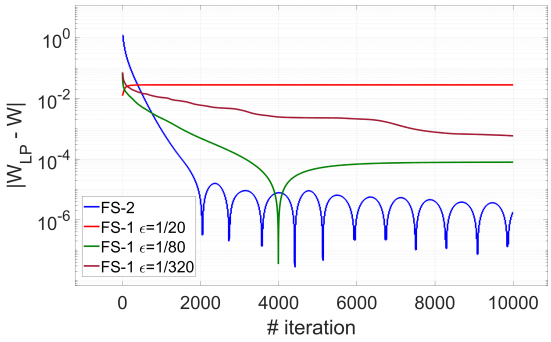

FIG. 3. *The 2D random distribution problem. The errors between the numerical results generated by FS-1 or FS-2 and the true Wasserstein-1 metric w.r.t. number of iterations.*

the $O(N^{4.98})$ complexity of the IPOT method. The computational time required to reach the absolute error of the Wasserstein-1 metric for $N \times N = 32 \times 32$ is also presented in Figure 4 (right). Similar to the previous subsection, we can also observe the huge computational efficiency of the FS-2 algorithm over the IPOT method.

6.3. Image matching problem. The final experiment tests the performance of our FS-2 algorithm for high-resolution image matching. This is a successful application of the Optimal Transport [4, 17, 7]. We select two images from the DIV2K dataset [1]. Through a process similar to Subsection 5.4 in the manuscript [24], we compute the Wasserstein-1 metric between the two images. The differences in the Wasserstein-1 metric between the true solution W_{LP} and the numerical solutions generated by FS-1 and FS-2 are depicted in Figure 6. We also present the averaged computational time of the IPOT method and the FS-2 algorithm in Table 3. Moreover, the computational time required to reach the absolute error of the Wasserstein-1 metric for $N \times N = 32 \times 32$ is shown in Figure 7. From these results, we can get the same conclusion as before, that is, the FS-2 algorithm seems to be the numerical algorithm with the fastest convergence and the lowest complexity for computing the Wasserstein-1 metric.

7. Conclusion. As the follow-up of the FS-1 paper, we generalize the result of matrix-vector multiplication at $O(N)$ costs for the special matrix to the more general L/U-CoLT matrix. We illustrate that only two vectors are required to represent any L/U-CoLT matrix. Moreover, we also prove the closure of families of L/U-CoLT matrices to matrix Hadamard product and matrix scaling. Therefore, the above

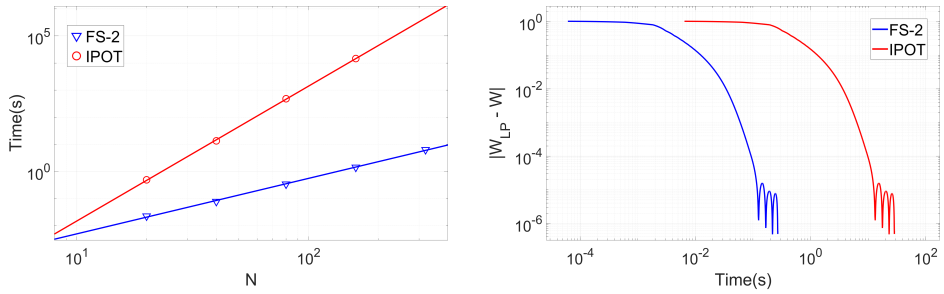

FIG. 4. *The 2D random distribution problem. Left: The comparison of computational time between the FS-2 algorithm and the IPOT method with different numbers of grid points N. Right: The computational time required to reach the absolute error of the Wasserstein-1 metric.*

TABLE 2

The 2D random distribution problem. The comparison between the IPOT method and the FS-2 algorithm with different total number of grid nodes $N \times N$. Columns 2-4 are the averaged computational time of the two algorithms and the speed-up ratio of the FS-2 algorithm. Column 5 is the Frobenius norm of the difference between the transport plan computed by the two algorithms.

N×N	Computational time (s)		Speed-up ratio	$\|P_{FS} - P\|_F$
	FS-2	IPOT		
20×20	2.24×10^{-2}	4.88×10^{-1}	2.18×10^{1}	2.40×10^{-16}
40×40	7.74×10^{-2}	1.34×10^{1}	1.73×10^{2}	1.52×10^{-16}
80×80	3.38×10^{-1}	4.79×10^{2}	1.42×10^{3}	1.40×10^{-16}
160×160	1.42×10^{0}	1.46×10^{4}	1.03×10^{4}	8.51×10^{-17}
320×320	6.31×10^{0}	–	–	–

FIG. 5. *The image matching problem. Illustration of images.*

TABLE 3

The image matching problem. The comparison between the IPOT method and the FS-2 algorithm with the different total number of grid nodes $N \times N$. Columns 2-4 are the averaged computational time of the two algorithms and the speed-up ratio of the FS-2 algorithm. Column 5 is the Frobenius norm of the difference between the transport plan computed by the two algorithms.

$N \times N$	Computational time (s)		Speed-up ratio	$\|P_{FS} - P\|_F$
	FS-2	IPOT		
100×100	5.16×10^{-1}	1.44×10^{3}	2.79×10^{3}	6.44×10^{-17}
200×200	2.31×10^{0}	3.69×10^{4}	1.60×10^{4}	4.65×10^{-17}
400×400	9.69×10^{0}	–	–	–
800×800	4.18×10^{1}	–	–	–

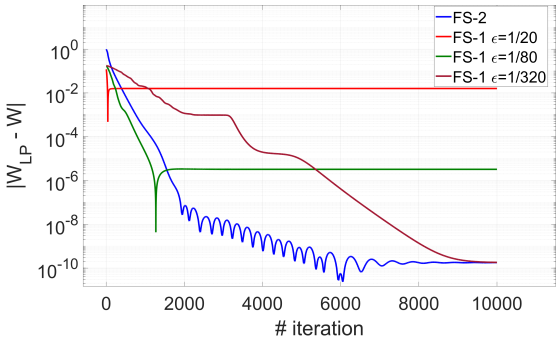

FIG. 6. *The image matching problem. The errors between the numerical results generated by FS-1 or FS-2 and the true Wasserstein-1 metric w.r.t. number of iterations.*

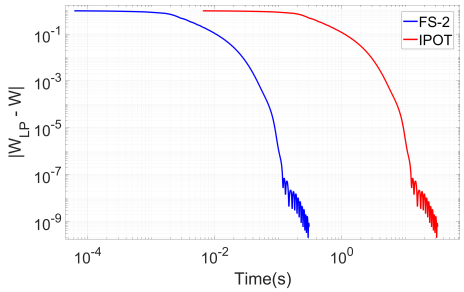

FIG. 7. *The image matching problem. The computational time required to reach the absolute error of the Wasserstein-1 metric.*

matrix operations are essentially updating the representation vectors, which reduce both time and space complexity to $O(N)$. These results can be directly applied to the Inexact Proximal point method for Optimal Transport problem and reduce the overall computational complexity to O(N). From this, we develop the Fast Sinkhorn II algorithm. It does not seem to be an overstatement that for the computation of the Wasserstein-1 metric, we have probably obtained the most competitive method, both in terms of convergence speed and computational complexity.

REFERENCES

[1] E. AGUSTSSON AND R. TIMOFTE, *Ntire 2017 challenge on single image super-resolution: Dataset and study*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017.

[2] J.-D. BENAMOU AND Y. BRENIER, *A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem*, Numer. Math., 84 (2000), pp. 375–393.

[3] J.-D. BENAMOU, B. D. FROESE, AND A. M. OBERMAN, *Numerical solution of the optimal transportation problem using the Monge–Ampère equation*, J. Comput. Phys., 260 (2014), pp. 107–126.

[4] M. BURGER, M. FRANEK, AND C.-B. SCHÖNLIEB, *Regularized regression and density estimation based on optimal transport*, Appl. Math. Res. Express, 2012 (2012), pp. 209–253.

[5] G. BUTTAZZO, L. DE PASCALE, AND P. GORI-GIORGI, *Optimal-transport formulation of electronic density-functional theory*, Phys. Rev. A, 85 (2012), p. 062502.

[6] J. CHEN, Y. CHEN, H. WU, AND D. YANG, *The quadratic Wasserstein metric for earthquake location*, J. Comput. Phys., 373 (2018), pp. 188–209.

[7] P. CLARYSSE, B. DELHAY, M. PICQ, AND J. POUSIN, *Optimal extended optical flow subject to*

a statistical constraint, J. Comput. Appl. Math., 234 (2010), pp. 1291–1302.
[8] P. L. Combettes and J.-C. Pesquet, *Proximal splitting methods in signal processing*, in Fixed-Point Algorithms for Inverse Problems in Science and Engineering, Springer, 2011, pp. 185–212.
[9] C. Cotar, G. Friesecke, and C. Klüppelberg, *Density functional theory and optimal transportation with Coulomb cost*, Comm. Pure Appl. Math., 66 (2013), pp. 548–599.
[10] M. Cuturi, *Sinkhorn distances: Lightspeed computation of optimal transport*, in Advances in Neural Information Processing Systems, vol. 26, 2013, pp. 2292–2300.
[11] I. Dikin, *Iterative solution of problems of linear and quadratic programming*, Dokl. Akad. Nauk, 174 (1967), pp. 747–748.
[12] B. Engquist, K. Ren, and Y. Yang, *The quadratic Wasserstein metric for inverse data matching*, Inverse Problems, 36 (2020), p. 055001.
[13] J. Franklin and J. Lorenz, *On the scaling of multidimensional matrices*, Linear Algebra Appl., 114 (1989), pp. 717–735.
[14] B. D. Froese, *Numerical methods for the elliptic Monge-Ampère equation and optimal transport*, PhD thesis, Simon Fraser University, Burnaby, BC, Canada, 2012.
[15] B. D. Froese and A. M. Oberman, *Convergent finite difference solvers for viscosity solutions of the elliptic Monge–Ampère equation in dimensions two and higher*, SIAM J. Numer. Anal., 49 (2011), pp. 1692–1714.
[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in Advances in Neural Information Processing Systems, vol. 27, 2014.
[17] S. Haker, L. Zhu, A. Tannenbaum, and S. Angenent, *Optimal mass transport for registration and warping*, Int. J. Comput. Vis., 60 (2004), pp. 225–240.
[18] H. Heaton, S. W. Fung, A. T. Lin, S. Osher, and W. Yin, *Wasserstein-based projections with applications to inverse problems*, arXiv preprint arXiv:2008.02200, (2020).
[19] Y. Hu, H. Chen, and X. Liu, *A global optimization approach for multi-marginal optimal transport problems with Coulomb cost*, arXiv preprint arXiv:2110.07352, (2021).
[20] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, in Proceedings of the sixteenth annual ACM symposium on Theory of computing, 1984, pp. 302–311.
[21] W. Li, E. K. Ryu, S. Osher, W. Yin, and W. Gangbo, *A parallel method for earth mover's distance*, J. Sci. Comput., 75 (2018), pp. 182–197.
[22] X. Li, D. Sun, and K.-C. Toh, *An asymptotically superlinearly convergent semismooth Newton augmented Lagrangian method for linear programming*, SIAM J. Optim., 30 (2020), pp. 2410–2440.
[23] Z. Li, Y. Tang, J. Chen, and H. Wu, *The quadratic Wasserstein metric with squaring scaling for seismic velocity inversion*, arXiv preprint arXiv:2201.11305, (2022).
[24] Q. Liao, J. Chen, Z. Wang, B. Bai, S. Jin, and H. Wu, *Fast Sinkhorn I: An $O(N)$ algorithm for the Wasserstein-1 metric*, Commun. Math. Sci., accepted, (2022).
[25] A. T. Lin, W. Li, S. Osher, and G. Montúfar, *Wasserstein proximal of GANs*, in International Conference on Geometric Science of Information, Springer, 2021, pp. 524–533.
[26] C. Meng, Y. Ke, J. Zhang, M. Zhang, W. Zhong, and P. Ma, *Large-scale optimal transport map estimation using projection pursuit*, in Advances in Neural Information Processing Systems, vol. 32, 2019, pp. 8118–8129.
[27] C. Meng, J. Yu, J. Zhang, P. Ma, and W. Zhong, *Sufficient dimension reduction for classification using principal optimal transport direction*, in Advances in Neural Information Processing Systems, vol. 33, 2020.
[28] L. Métivier, R. Brossier, Q. Merigot, É. Oudet, and J. Virieux, *An optimal transport approach for seismic tomography: Application to 3D full waveform inversion*, Inverse Problems, 32 (2016), p. 115008.
[29] O. Museyko, M. Stiglmayr, K. Klamroth, and G. Leugering, *On the application of the Monge–Kantorovich problem to image registration*, SIAM J. Imaging Sci., 2 (2009), pp. 1068–1097.
[30] O. Pele and M. Werman, *Fast and robust earth mover's distances*, in 2009 IEEE 12th International Conference on Computer Vision, IEEE, 2009, pp. 460–467.
[31] G. Peyré, M. Cuturi, et al., *Computational optimal transport: With applications to data science*, Found. Trends Mach. Learn., 11 (2019), pp. 355–607.
[32] Y. Rubner, C. Tomasi, and L. J. Guibas, *The earth mover's distance as a metric for image retrieval*, Int. J. Comput. Vis., 40 (2000), pp. 99–121.
[33] F. Santambrogio, *Optimal transport for applied mathematicians: Calculus of variations, pdes, and modeling*, Progr. Nonlinear Differential Equations Appl., Birkäuser, Basel, 2015.
[34] R. Sinkhorn, *Diagonal equivalence to matrices with prescribed row and column sums*, Amer.

445 Math. Monthly, 74 (1967), pp. 402–405.
446 [35] Y. Xie, X. Wang, R. Wang, and H. Zha, *A fast proximal point method for computing exact*
447 *Wasserstein distance*, in Uncertainty in artificial intelligence, PMLR, 2020, pp. 433–453.
448 [36] L. Yang, J. Li, D. Sun, and K.-C. Toh, *A fast globally linearly convergent algorithm for the*
449 *computation of Wasserstein barycenters.*, J. Mach. Learn. Res., 22 (2021), pp. 1–37.
450 [37] Y. Yang, B. Engquist, J. Sun, and B. F. Hamfeldt, *Application of optimal transport and the*
451 *quadratic Wasserstein metric to full-waveform inversion*, Geophysics, 83 (2018), pp. R43–
452 R62.