# Interior-point Methods Strike Back: Solving the Wasserstein Barycenter Problem

**Dongdong Ge**
Research Institute for Interdisciplinary Sciences
Shanghai University of Finance and Economics
ge.dongdong@mail.shufe.edu.cn

**Haoyue Wang***
School of Mathematical Sciences
Fudan University
haoyuewang14@fudan.edu.cn

**Zikai Xiong***
School of Mathematical Sciences
Fudan University
zkxiong16@fudan.edu.cn

**Yinyu Ye**
Department of Management Science and Engineering
Stanford University
yyye@stanford.edu

## Abstract

Computing the Wasserstein barycenter of a set of probability measures under the optimal transport metric can quickly become prohibitive for traditional second-order algorithms, such as interior-point methods, as the support size of the measures increases. In this paper, we overcome the difficulty by developing a new adapted interior-point method that fully exploits the problem's special matrix structure to reduce the iteration complexity and speed up the Newton procedure. Different from regularization approaches, our method achieves a well-balanced tradeoff between accuracy and speed. A numerical comparison on various distributions with existing algorithms exhibits the computational advantages of our approach. Moreover, we demonstrate the practicality of our algorithm on image benchmark problems including MNIST and Fashion-MNIST.

## 1 Introduction

To compare, summarize, and combine probability measures defined on a space is a fundamental task in statistics and machine learning. Given support points of probability measures in a metric space and a transportation cost function (e.g. the Euclidean distance), Wasserstein distance defines a distance between two measures as the minimal transportation cost between them. This notion of distance leads to a host of important applications, including text classification [30], clustering [25, 26, 15, 31], unsupervised learning [23, 13], semi-supervised learning [47], supervised-learning[27, 19], statistics [38, 39, 48, 21], and others [7, 41, 1, 44, 37]. Given a set of measures in the same space, the 2-Wasserstein barycenter is defined as the measure minimizing the sum of squared 2-Wasserstein distances to all measures in the set. For example, if a set of images (with common structure but varying noise) are modeled as probability measures, then the Wasserstein barycenter is a mixture of the images that share this common structure. The Wasserstein barycenter better captures the underlying geometric structure than the barycenter defined by the Euclidean or other distances. As a result, the Wasserstein barycenter has applications in clustering [25, 26, 15], image retrieval [14] and others [32, 43, 11, 34].

From the computation point of view, finding the barycenter of a set of discrete measures can be formulated by linear programming[6, 8]. Nonetheless, state-of-the-art linear programming solvers do not scale with the immense amount of data involved in barycenter calculations. Current research

*Haoyue Wang and Zikai Xiong are corresponding authors.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

on computation mainly follows two types of methods. The first type attempts to solve the linear program (or some equivalent problem) with scalable first-order methods. J.Ye et al. [54] use modified Bregman ADMM(BADMM) – introduced by [51] – to compute Wasserestein barycenters for clustering problems. L.Yang et al. [53] adopt symmetric Gauss-Seidel ADMM to solve the dual linear program, which reduces the computational cost in each iteration. S.Claici et al. [12] introduce a stochastic alternating algorithm that can handle continuous input measures. However, these methods are still computationally inefficient when the number of support points of the input measures and the number of input measures are large. Due to the nature of the first-order methods, these algorithms often converge too slowly to reach high-accuracy solutions.

The second, more mainstream, approach introduces an entropy regularization term to the linear programming formulation[14, 9]. This technique was first developed in solving optimal transportation problem. See [14, 3, 18, 50, 33, 10, 22] for the related works. M. Staib et al. [49] discuss the parallel computation issue and introduce a sampling method. P.Dvurechenskii et al. [17] study decentralized and distributed computation for the regularized problem. These methods are indeed suitable for large-scale problems due to their low computational cost and parsimonious memory usage. However, this advantage is obtained at the expense of the solution accuracy: especially when the regularization term is weighted less in order to approximate the original problem more accurately, computational efficiency degenerates and the outputs become unstable [9]. S. Amari et al. [5] propose a entropic regularization based sharpening technique but their result is not the accurate real barycenter. P.C. Alvarez-Esteban et al. [4] prove that the barycenter must be the fixed-point of a new operator. See [42] for a detailed survey of related algorithms.

In this paper, we develop a new interior-point method (IPM), namely Matrix-based Adaptive Alternating interior-point Method (MAAIPM), to efficiently calculate the Wasserstein barycenters. If the support is pre-specified, we apply one step of the Mizuno-Todd-Ye predictor-corrector IPM[36]. The algorithm gains a quadratic convergence rate showed by Y. Ye et al. [56], which is a distinct advantage of IPMs over first-order methods. In practice, we implement Mehrotra's predictor-corrector IPM [35], and add clever heuristics in choosing step lengths and centering parameters. If the support is also to be optimized, MAAIPM alternatively updates support and linear program variables in an adaptive strategy. At the beginning, MAAIPM updates support points $X^*$ by an unconstrained quadratic program after a few number of IPM iterations. At the end, MAAIPM updates $X^*$ after every IPM iteration and applies the "jump" tricks to escape local minima. Under the framework of MAAIPM, we present two block matrix-based accelerated algorithms to quickly solve the Newton equations at each iteration. Despite a prevailing belief that IPMs are inefficient for large-scale cases, we show that such an inefficiency can be overcome through careful manipulation of the block-data structure of the normal equation. As a result, our stylized IPM has the following advantages.

**Low theoretical complexity.** The linear programming formulation of the Wasserstein barycenter has $m\sum_{i=1}^{N} m_i + m$ variables andred$Nm + \sum_{i=1}^{N} m_i + 1$ constraints, where the integers $N$, $m$ and $m_i$ will be specified later. Although MAAIPM is still a second-order method, in our two block matrix-based accelerated algorithms, every iteration of solving the Newton direction has a time complexity of merely $O(m^2\sum_{i=1}^{N} m_i + Nm^3)$ or $O(m\sum_{i=1}^{N} m_i^2 + \sum_{i=1}^{N} m_i^3)$, where a standard IPM would need $O\big((Nm + \sum_{i=1}^{N} m_i + 1)^2(m\sum_{i=1}^{N} m_i + m)\big)$. For simplicity, let $m_i = m,\ i = 1, 2 \ldots, N$, then the time complexity of our algorithm in each iteration is $O(Nm^3)$, instead of standard IPM's complexity $O(N^3m^4)$. Note that theoretically, when $Nm^2 = (Nm)^k$ for some $1 < k < 2$, the complexity of the standard IPM can be reduced to $O((Nm)^{\omega(k)}) + O((Nm)^3)$ via fast matrix computation methods, where the specific value of $\omega(k)$ can be found in table 3 of [20])

**Practical effectiveness in speed and accuracy.** Compared to regularized methods, IPMs gain high-accuracy solutions and high convergence rate by nature. Numerical experiments show that our algorithm converges to highly accurate solutions of the original linear program with the least number of iterations. Figure 1 shows the advantages of our methods in accuracy in comparison to the well-developed Sinkhorn-type algorithm [14, 9].

Figure 1: A comparison of algorithms for computing the barycenters between a Sinkhorn based approach[9](left) and MAAIPM(right). Samples of handbag(first 4 rows) are from Fashion-MNIST dataset.

There are more advantages of our approaches in real implementation. When the support points of measures are the same, there are several specially designed highly memory-efficient and thus very fast Sinkhorn based algorithms such as [46, 9]. However, when the support points of measures are different, the convolutional method in [46] is no longer applicable and the memory usage of our method is within a constant multiple of the popular memory-efficient first-order Sinkhorn method, IBP[9], much less than the memory used by a commercial solver. In this case, experiments also show that our algorithm can perform the best in both accuracy and overall runtime. Our algorithms also inherits a natural structure potentially fitting parallel computing scheme well. Those merits ensure that our algorithm is highly suitable for large-scale computation of Wasserstein barycenters.

The rest of the paper is organized as follows. In section 2, we briefly define the Wasserstein barycenter. In section 3, we present its linear programming formulation and introduce the IPM framework. In section 4, we present an IPM implementation that greatly reduces the computational cost of classical IPMs. In section 5, we present our numerical results.

## 2 Background and Preliminaries

In this section, we briefly recall the Wasserstein distance and the Wasserstein barycenter for a set of discrete probability measures [2, 16]. Let $\Sigma_n = \{\boldsymbol{a} \in \mathbb{R}^n | \sum_{i=1}^n a_i = 1, a_i \geq 0 \text{ for } i = 1, 2, \dots n\}$ be the probability simplex in $\mathbb{R}^n$. For two vectors $\boldsymbol{s}^{(1)} \in \Sigma_{n_1}, \boldsymbol{s}^{(2)} \in \Sigma_{n_2}$, define the set of matrices $\mathcal{M}(\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}) = \{\Pi \in \mathbb{R}_+^{n_1 \times n_2} : \Pi \mathbf{1}_{n_2} = \boldsymbol{s}^{(1)}, \Pi^\top \mathbf{1}_{n_1} = \boldsymbol{s}^{(2)}\}$. Let $\mathcal{P} = \{(a_i, \boldsymbol{q}_i) : i = 1, \dots, m\}$ denote the discrete probability measure supported on $m$ points $\boldsymbol{q}_1, \dots, \boldsymbol{q}_m$ in $\mathbb{R}^d$ with weights $a_1, \dots, a_m$ respectively. The Wasserstein barycenter of the two measures $\mathcal{U} = \{(a_i, \boldsymbol{q}_i) : i = 1, \dots, m_1\}$ and $\mathcal{V} = \{(b_j, \boldsymbol{p}_j) : j = 1, \dots, m_2\}$ is

$$\mathcal{W}_2(\mathcal{U}, \mathcal{V}) := \min \left\{ \sqrt{\sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \pi_{ij} \|\boldsymbol{q}_i - \boldsymbol{p}_j\|^2} : \Pi = [\pi_{ij}] \in \mathcal{M}(\boldsymbol{a}, \boldsymbol{b}) \right\} \tag{1}$$

where $\boldsymbol{a} = (a_1, \dots, a_{m_1})^\top$ and $\boldsymbol{b} = (b_1, \dots, b_{m_2})^\top$. Consider a set of probability measures $\{\mathcal{P}^{(t)}, t = 1, \cdots, N\}$ where $\mathcal{P}^{(t)} = \{(a_i^{(t)}, \boldsymbol{q}_i^{(t)}) : i = 1, \dots, m_t\}$, and let $\boldsymbol{a}^{(t)} = (a_1^{(t)}, \dots, a_{m_t}^{(t)})^\top$. The Wasserstein barycenter (with $m$ support points) $\mathcal{P} = \{(w_i, \boldsymbol{x}_i) : i = 1, \cdots, m\}$ is another probability measure which is defined as a solution of the problem

$$\min_{\mathcal{P}} \frac{1}{N} \sum_{t=1}^{N} (\mathcal{W}_2(\mathcal{P}, \mathcal{P}^{(t)}))^2. \tag{2}$$

Furthermore, define the simplex $\mathcal{S} = \big\{(\boldsymbol{w}, \Pi^{(1)}, \dots, \Pi^{(N)}) \in \mathbb{R}_+^m \times \mathbb{R}_+^{m \times m_1} \times \cdots \times \mathbb{R}_+^{m \times m_N} : \mathbf{1}_m^\top \boldsymbol{w} = 1, \boldsymbol{w} \geq 0;\ \Pi^{(t)} \mathbf{1}_{m_t} = \boldsymbol{w}, \left(\Pi^{(t)}\right)^\top \mathbf{1}_m = \boldsymbol{a}^{(t)}, \Pi^{(t)} \geq 0, \forall t = 1, \cdots, N\big\}$. For a given set of support points $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$, define the distance matrices $D^{(t)}(X) = [\|\boldsymbol{x}_i - \boldsymbol{q}_j^{(t)}\|_2^2] \in \mathbb{R}^{m \times m_t}$ for $t = 1, \dots, N$. Then problem (2) is equivalent to

$$\min_{\boldsymbol{w}, X, \Pi^{(t)}} \ \sum_{t=1}^{N} \left\langle D^{(t)}(X), \Pi^{(t)} \right\rangle \ \text{ s.t. } (\boldsymbol{w}, \Pi^{(1)}, \dots, \Pi^{(N)}) \in \mathcal{S},\ \boldsymbol{x}_1, \dots, \boldsymbol{x}_m \in \mathbb{R}^n. \tag{3}$$

Problem (3) is a nonconvex problem, where one needs to find the optimal support points X and the optimal weight vector $\boldsymbol{w}$ of a barycenter simultaneously. However, in many real applications, the support X of a barycenter can be specified empirically from the support points of $\{\mathcal{P}^{(t)}\}_{t=1}^N$. Indeed, in some cases, all measures in $\{\mathcal{P}^{(t)}\}_{t=1}^N$ have the same set of support points and hence the barycenter should also take the same set of support points. In view of this, we will also focus on the case when the support X is given. Consequently, problem (3) reduces to the following problem:

$$\min_{\boldsymbol{w}, \Pi^{(t)}} \ \sum_{t=1}^{N} \left\langle D^{(t)}, \Pi^{(t)} \right\rangle \ \text{ s.t. } (\boldsymbol{w}, \Pi^{(1)}, \dots, \Pi^{(N)}) \in \mathcal{S} \tag{4}$$

where $D^{(t)}$ denotes $\mathcal{D}(X, Q^{(t)})$ for simplicity. In the following sections, we refer to problem (4) as the *Pre-specified Support Problem*, and call problem (3) the *Free Support Problem*.

## 3 General Framework for MAAIPM

**Linear programming formulation and preconditioning.** Note that the Pre-specified Support Problem is a linear program. In this subsection, we focus on removing redundant constraints. First, we vectorize the constraints $\Pi^{(t)}\mathbf{1}_{m_t} = \boldsymbol{w}$ and $\left(\Pi^{(t)}\right)^\top \mathbf{1}_m = \boldsymbol{a}^{(t)}$ captured in $\mathcal{S}$ to become

$$(\mathbf{1}_{m_t}^\top \otimes I_m)vec(\Pi^{(t)}) = \boldsymbol{w},\ \ (I_{m_t} \otimes \mathbf{1}_m^\top)vec(\Pi^{(t)}) = \boldsymbol{a}^{(t)},\ \ t = 1, \cdots, N.$$

Thus, problem (4) can be formulated into the standard-form linear program:

$$\min\ \boldsymbol{c}^\top \boldsymbol{x}\ \text{ s.t. }\ A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0 \tag{5}$$

with $\boldsymbol{x} = (vec(\Pi^{(1)}); ...; vec(\Pi^{(N)}); \boldsymbol{w})$ , $b = (\boldsymbol{a}^{(1)}; \boldsymbol{a}^{(2)}; ...; \boldsymbol{a}^{(N)}; \mathbf{0}_m; ...; \mathbf{0}_m; 1)$, $\boldsymbol{c} = (vec(D^{(1)}); ...; vec(D^{(N)}); \mathbf{0})$ and $A = \begin{bmatrix} E_1^\top & E_2^\top & 0 \\ 0 & E_3^\top & \mathbf{1}_m \end{bmatrix}^\top$, where $E_1$ is a block diagonal matrix: $E_1 = diag(I_{m_1} \otimes \mathbf{1}_m^\top, ..., I_{m_N} \otimes \mathbf{1}_m^\top)$; $E_2$ is a block diagonal matrix: $E_2 = diag(\mathbf{1}_{m_1}^\top \otimes I_m, ..., \mathbf{1}_{m_N}^\top \otimes I_m)$; and $E_3 = -\mathbf{1}_N \otimes I_m$. Let $M := \sum_{i=1}^N m_i$, $n_{row} := Nm + \sum_{i=1}^N m_i + 1$ and $n_{col} := m\sum_{i=1}^N m_i + m$. Then $A \in \mathbb{R}^{n_{row} \times n_{col}}$, $\boldsymbol{b} \in \mathbb{R}^{n_{row}}$ and $\boldsymbol{c} \in \mathbb{R}^{n_{col}}$. We are faced with a standard form linear program with $n_{col}$ variables and $n_{row}$ constraints. In the spacial case where all $m_i = m$, the number of variables is $O(Nm)$, and the number of constraints is $O(Nm^2)$.

For efficient implementations of IPMs for this linear program, we need to remove redundant constraints.

**Lemma 3.1** *Let $\bar{A} \in \mathbb{R}^{(n_{row}-N)\times n_{col}}$ be obtained from $A$ by removing the $(M+1)$-th, $(M+m+1)$-th, $\cdots$, $(M+(N-1)m+1)$-th rows of $A$, and $\bar{\boldsymbol{b}} \in \mathbb{R}^{n_{row}-N}$ be obtained from $\boldsymbol{b}$ by removing the $(M+1)$-th, $(M+m+1)$-th, $\cdots$, $(M+(N-1)m+1)$-th entries of $\boldsymbol{b}$. Then 1) $\bar{A}$ has full row rank; 2) $\boldsymbol{x}$ satisfies $A\boldsymbol{x} = \boldsymbol{b}$ if and only if $\boldsymbol{x}$ satisfies $\bar{A}\boldsymbol{x} = \bar{\boldsymbol{b}}$.*

The proof of this lemma is available in the appendix. With this lemma, the primal problem and dual problem of problem 5 can be written as

$$\text{(Primal)}\min\ \boldsymbol{c}^\top \boldsymbol{x}\ \text{ s.t. }\ \bar{A}\boldsymbol{x} = \bar{\boldsymbol{b}}, \boldsymbol{x} \geq 0. \quad \text{(Dual)}\max\ \bar{\boldsymbol{b}}^\top \boldsymbol{p}\ \text{ s.t. }\ \bar{A}^\top \boldsymbol{\lambda} + \boldsymbol{s} = \boldsymbol{c}, \boldsymbol{s} \geq 0. \tag{6}$$

**Framework of Matrix-based Adaptive Alternating Interior-point Method (MAAIPM).** When the support points are not pre-specified, we need to solve problem (3). As we just saw, When $X$ is fixed, the problem becomes a linear program. When $(\boldsymbol{w}, \{\Pi^{(t)}\})$ are fixed, the problem is a quadratic optimization problem with respect to $X$, and the optimal $X^*$ can be written in closed form as

$$\boldsymbol{x}_i^* = \left(\sum_{t=1}^N \sum_{j=1}^{m_t} \pi_{ij}^{(t)}\right)^{-1} \sum_{t=1}^N \sum_{j=1}^{m_t} \pi_{ij}^{(t)} \boldsymbol{q}_j^{(t)}, \quad i = 1, 2 \ldots, m. \tag{7}$$

In anther word, (3) can be reformulated as

$$\min \boldsymbol{c}(\boldsymbol{x})^\top \boldsymbol{x}\ \text{ s.t.}\bar{A}\boldsymbol{x} = \bar{\boldsymbol{b}},\ \boldsymbol{x} \geq 0. \tag{8}$$

Since, as stated above, (3) is a non-convex problem and so it contains saddle points and local minima. This makes finding a global optimizer difficult. Examples of local minima and saddle points are available in the appendix. The alternating minimization strategy used in [16, 53, 54] alternates between optimizing $X$ by solving (7) and optimizing $(\boldsymbol{w}, \{\Pi^{(t)}\})$ by solving (4). However, this alternating approach cannot avoid local minima or saddle points. Every iteration may require solving a linear program (4), which is expensive when the problem size is large.

To overcome the drawbacks, we propose Matrix-based Adaptive Alternating IPM (MAAIPM). If the support is pre-specified, we solve a single linear program by predictor-corrector IPM[35, 40, 52]. If the support should be optimized, MAAIPM uses an adaptive strategy. At the beginning, because the primal variables are far from the optimal solution, MAAIPM updates $X^*$ of (7) after a few number of IPM iterations for $(\boldsymbol{w}, \{\Pi^{(t)}\})$. Then, MAAIPM updates $X^*$ after every IPM iteration and applies the "jump" tricks to escape local minima. Although MAAIPM cannot ensure finding a globally optimal solution, it can frequently get a better solution in shorter time. Since at the beginning MAAIPM updates $X^*$ after many IPM iterations, primal dual predictor-corrector IPM is more efficient. At the end, $X^*$ is updated more often and each update of $X^*$ changes the linear programming objective

function so that dual variables may be infeasible. However, the primal variables always remain feasible so that the primal IPM is more suitable at the end. Moreover, primal IPM is better for applying "jump" tricks or other local-minima-escaping techniques, which has been shown in [55]. Details and illustration are available in the appendix.

In predictor-corrector IPM, the main computational cost lies in solving the Newton equations, which can be reformulated as the normal equations

$$\bar{A}(D^k)^2\bar{A}^\top\Delta\boldsymbol{\lambda}^k = \boldsymbol{f}^k, \tag{9}$$

where $D^k$ denotes $diag(x_i^{(k)}/s_i^{(k)})$ and $\boldsymbol{f}^k$ is in $\mathbb{R}^{n_{row}-N}$. This linear system of matrix $\bar{A}(D^k)^2\bar{A}^\top$ can be efficiently solved by the two methods proposed in the next section. In the primal IPM, MAAIPM combines following the central path with optimizing the support points, i.e., it contains three parts in one iteration, taking an Newton step in the logarithmic barrier function

$$\text{minimize} \quad \boldsymbol{c}^\top\boldsymbol{x} - \mu\textstyle\sum_{i=1}^n \ln x_i, \quad \text{subject to} \quad \bar{A}\boldsymbol{x} = \boldsymbol{b}, \tag{10}$$

reducing the penalty $\mu$, and updating the support (7). The Newton direction $\boldsymbol{p}_k$ at the $k^{th}$ iteration is calculated by

$$\boldsymbol{p}^k = \boldsymbol{x}^k + (X^k)^2\Big(\bar{A}^\top\big(\bar{A}(X^k)^2\bar{A}^\top\big)^{-1}\big(\bar{A}(X^k)^2\boldsymbol{c} - \mu\bar{A}X^k\mathbf{1}\big) - \boldsymbol{c}\Big)/\mu^k, \tag{11}$$

where $X^k = diag(x_i^{(k)})$. The main cost of primal IPM lies in solving a linear system of $\bar{A}(X^k)^2\bar{A}^\top$, which again can be efficiently solved by the two methods described in the following section. Further more, we also apply the warm-start technique to smartly choose the starting point of the next IPM after "jump" [45]. Compared with primal-dual IPMs' warm-start strategies [29, 28], our technique saves the searching time, and only requires slightly more memory. When we suitably set the termination criterion, numerical studies show that MAAIPM outperforms previous algorithms in both speed and accuracy, no matter whether the support is pre-specified or not.

## 4 Efficient Methods for Solving the Normal Equations

In this section, we discuss efficient methods for solving normal equations in the format $(\bar{A}D\bar{A}^\top)\boldsymbol{z} = \boldsymbol{f}$, where $D$ is a diagonal matrix with all diagonal entries being positive. Let $\boldsymbol{d} = diag(D)$, and $M_2 = N(m-1)$. First, through simple calculation, we have the following lemma on the structure of matrix $\bar{A}D\bar{A}^\top$, whose proof is available in the appendix.

**Lemma 4.1** *$\bar{A}D\bar{A}^T$ can be written in the following format:*

$$\bar{A}D\bar{A}^T = \begin{bmatrix} B_1 & B_2 & \mathbf{0} \\ B_2^\top & B_3 + B_4 & \boldsymbol{\alpha} \\ \mathbf{0} & \boldsymbol{\alpha}^\top & c \end{bmatrix}$$

*where $B_1 \in \mathbb{R}^{M\times M}$ is a diagonal matrix with positive diagonal entries; $B_2 \in \mathbb{R}^{M\times M_2}$ is a block-diagonal matrix with N blocks (the size of the i-th block is $(m-1)\times m_i$); $B_3 \in \mathbb{R}^{M_2\times M_2}$ is a diagonal matrix with positive diagonal entries; Let $\boldsymbol{y} = \boldsymbol{d}(n_{col}-m+2 : n_{col})$, then $B_4 = (\mathbf{1}_N\mathbf{1}_N^\top)\otimes diag(\boldsymbol{y})$, and $\boldsymbol{\alpha} = -\mathbf{1}_N \otimes \boldsymbol{y}$; $c = \mathbf{1}_m^\top\boldsymbol{d}(n_{col}-m+1 : n_{col})$.*

**Single low-rank regularization method (SLRM).** Briefly speaking, we will perform several basic transformations on the matrix $\bar{A}D\bar{A}^T$ to transform it into an easy-to-solve format. Then we solve the system with the transformed coefficient matrix and finally transform the obtained solution back to get an solution of $(\bar{A}D\bar{A}^\top)\boldsymbol{z} = \boldsymbol{f}$.

Define $V_1 := \begin{bmatrix} I_M & & \\ -B_2^\top B_1^{-1} & I_{M_2} & \\ & & 1 \end{bmatrix}$, $V_2 := \begin{bmatrix} I_M & & -\boldsymbol{\alpha}/c \\ & I_{M_2} & \\ & & 1 \end{bmatrix}$, $A_1 := B_3 - B_2^\top B_1^{-1}B_2$ and $A_2 := B_4 - \frac{1}{c}\boldsymbol{\alpha}\boldsymbol{\alpha}^\top$. Then,

$$V_2V_1\bar{A}D\bar{A}^TV_1^\top V_2^\top = \begin{bmatrix} B_1 & & \\ & B_3 - B_2^\top B_1^{-1}B_2 + B_4 - \frac{1}{c}\boldsymbol{\alpha}\boldsymbol{\alpha}^\top & \\ & & c \end{bmatrix} = \begin{bmatrix} B_1 & & \\ & A_1 + A_2 & \\ & & c \end{bmatrix}.$$

Define $Y = diag(\boldsymbol{y}) - \frac{1}{c}\boldsymbol{y}\boldsymbol{y}^\top$, we have the following lemma.

**Lemma 4.2**

***a)*** *$A_1$ is a block-diagonal matrix with N blocks. The size of each block is $(m-1)\times(m-1)$. Further more, $A_1$ is positive definite and strictly diagonal dominant.* ***b)*** *$A_2 = (\mathbf{1}_N\mathbf{1}_N^\top)\otimes Y$, and $Y$ is positive definite and strictly diagonal dominant.*

Since the positive definiteness and diagonal dominance claimed in this lemma, the computation of the inverse matrices of each block of $A_1$ and $A_2$ is numerically stable. Now we introduce the procedure for solving $(\bar{A}D\bar{A}^T)\boldsymbol{z} = \boldsymbol{f}$, as descried in Algorithm 1 ($z^{(1)}$ - $z^{(4)}$ in the algorithm are intermediate variables). In step 7, we need to solve a linear system with coefficient matrix of dimension $N(m-1)\times N(m-1)$, which is hard to compute with common methods for dense symmetric matrices. In view of the low-rank structure of the matrix $A_2$, we introduce a method, namely Single Low-rank Regularization Method (SLRM), which requires only $O(Nm^3)$ flops in computation. Assume $A_1 = diag(A_{11}, A_{22}, ..., A_{NN})$ and define $U = \begin{bmatrix} I_{N-1} & \mathbf{1}_{N-1} \\ 0 & 1 \end{bmatrix} \otimes I_{m-1}$. We can solve the linear system $(A_1 + A_2)\boldsymbol{x} = \boldsymbol{g}$ by Algorithm 2.

**Algorithm 1:** Solver for the normal equation $(\bar{A}D\bar{A}^T)\boldsymbol{z} = \boldsymbol{f}$

**Input:** $\boldsymbol{d} = diag(D) \in \mathbb{R}^{n_{col}}$; $\boldsymbol{f} \in \mathbb{R}^{M+N(m-1)+1}$

1 compute $B_1, B_2, B_3$, vector $\boldsymbol{y} = \boldsymbol{d}(n_{col} - m + 2 : n_{col})$ and $c$;
2 compute $T = B_2^\top B_1^{-1}$ and matrices $V_1, V_2$;
3 compute $A_1 = B_3 - TB_2$ and $A_2 = (\mathbf{1}_N\mathbf{1}_N^\top)\otimes(diag(\boldsymbol{y}) - \frac{1}{c}\boldsymbol{y}\boldsymbol{y}^\top)$;
4 compute $\boldsymbol{z}^{(1)} = V_1\boldsymbol{f}$ and $\boldsymbol{z}^{(2)} = V_2\boldsymbol{z}^{(1)}$;
5 compute $\boldsymbol{z}^{(3)}(1:M) = B_1^{-1}\boldsymbol{z}^{(2)}(1:M)$;
6 compute $\boldsymbol{z}^{(3)}(M+M_2+1) = \frac{1}{c}\boldsymbol{z}^{(2)}(M+M_2+1)$;
7 solve the linear system with coefficient matrix $A_1 + A_2$ to get $\boldsymbol{z}^{(3)}(M+1:M+M_2) = (A_1+A_2)^{-1}\boldsymbol{z}^{(2)}(M+1:M+M_2)$;
8 compute $\boldsymbol{z}^{(4)} = V_2^\top\boldsymbol{z}^{(3)}$ , $\boldsymbol{z} = V_1^\top\boldsymbol{z}^{(4)}$;

**Output:** $\boldsymbol{z}$

The proof of correctness of Algorithm 2 and other analysis is available in the appendix.

**Algorithm 2:** SLRM for the system $(A_1 + A_2)\boldsymbol{x} = \boldsymbol{g}$

**Input:** $A_1,\ A_2,\ \boldsymbol{g}$

1 compute $A_{ii}^{-1}, i = 1, .., N$;
2 set $A_1^{-1} = diag(A_{11}^{-1}, ..., A_{NN}^{-1})$;
3 compute $\boldsymbol{x}^{(1)} = A_1^{-1}\boldsymbol{g}$;
4 compute $\boldsymbol{x}^{(2)} = U^T\boldsymbol{x}^{(1)}$;
5 compute $\boldsymbol{x}^{(3)}(end-m+2:end) = (Y^{-1} + \sum_{i=1}^N A_{ii}^{-1})\backslash\boldsymbol{x}^{(2)}(end-m+2:end)$;
6 set $\boldsymbol{x}^{(3)}(1:end-m+1) = 0$ ;
7 compute $\boldsymbol{x}^{(4)} = U\boldsymbol{x}^{(3)}$ and $\boldsymbol{x}^{(5)} = A_1^{-1}\boldsymbol{x}^{(4)}$;
8 compute $\boldsymbol{x} = \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(5)}$;

**Output:** $\boldsymbol{x}$

**Double low-rank regularization method (DLRM) when $m$ is large.** In many applications, $m$ is relatively large compared to $m_t$. For instance, in the area of image identification, the pixel support points of the images at hand are sparse (small $m_t$) but different. To find the "barycenter" of these images, we need to assume the "barycenter" image has much more pixel support points (large $m$) than all the sample images. Sometimes, $m$ might be about 5 to 20 times of each $m_t$. In this case, the computational cost of step 1 in SLRM is heavy, since we need to solve $N$ linear systems with dimension $m \times m$. In this subsection, we use the low rank regularization formula to further reduce the computational cost.

In view of lemma 4.1, assume

$$B_1 = diag(B_{11}, ..., B_{1N}),\ \ B_2 = diag(B_{21}, ..., B_{2N}),\ \ B_3 = diag(B_{31}, ..., B_{3N}).$$

where $B_{1i} \in \mathbb{R}^{m_i\times m_i}$, $B_{2i} \in \mathbb{R}^{m_i\times(m-1)}$ and $B_{3i} \in \mathbb{R}^{(m-1)\times(m-1)}$. Recall that $A_1 = B_3 - B_2^\top B_1^{-1}B_2$ and $A_1 = diag(A_{11}, ..., A_{NN})$, we have $A_{ii} = B_{3i} - B_{2i}^\top B_{1i}^{-1}B_{2i}$. Since $m >> m_i$, we can use the following formula:

$$A_{ii}^{-1} = (B_{3i} - B_{2i}^\top B_{1i}^{-1}B_{2i})^{-1} = B_{3i}^{-1} + B_{3i}^{-1}B_{2i}^\top(B_{1i} - B_{2i}B_{3i}^{-1}B_{2i}^\top)^{-1}B_{2i}B_{3i}^{-1}. \tag{12}$$

Instead of calculating and storing each $A_{ii}$ explicitly, we can just calculate and store each $(B_{1i} - B_{2i}B_{3i}^{-1}B_{2i}^\top)^{-1}$. When we need to calculate $A_{ii}\boldsymbol{y}$ for some vector $\boldsymbol{y}$, we can use (12) and sequentially

multiply each matrix with vectors. As a result, the flops required in step 1 of SLRM reduce to $O(m\Sigma_{i=1}^{N} m_i^2 + \Sigma_{i=1}^{N} m_i^3)$, and the total memory usage of whole MAAIPM is $O(m\Sigma_{i=1}^{N} m_i)$, which is at the same level (except for a constant) of a primal variable.

**Complexity analysis.** The following theorem summarizes the time and space complexity of the aforementioned two methods.

**Theorem 4.3** ***a)*** *For SLRM, the time complexity in terms of flops is* $O(m^2 \sum_{i=1}^{N} m_i + Nm^3)$*, and the memory usage in terms of doubles is* $O(m \sum_{i=1}^{N} m_i + Nm^2)$*;* ***b)*** *For the DLRM, the time complexity in terms of flops is* $O(m \sum_{i=1}^{N} m_i^2 + \sum_{i=1}^{N} m_i^3)$*, and the memory usage in terms of doubles is* $O(m \sum_{i=1}^{N} m_i + \sum_{i=1}^{N} m_i^2)$*.*

We can choose between SLRM and DLRM for different cases to achieve lower time and space complexity. Note that as $N, m, m_i$ grows up, the memory usage here is within an constant time of the representative Sinkhorn type algorithms like IBP[9].

## 5 Experiments

We conduct three numerical experiments to investigate the real performance of our methods. The first experiment shows the advantages of SLRM and DLRM over traditional approaches in solving Newton equations with a same structure as barycenter problems. The second experiment fully demonstrates the merits of MAAIPM: high speed/accuracy and more efficient memory usage. In the last experiment with real benchmark data, MAAIPM recovers the images better than any other approach implemented. In different experiments, we compare our methods with state-of-art commercial solvers(MATLAB, Gurobi, MOSEK), the iterative Bregman projection (IBP) by [9], Bregman ADMM (BADMM) [51, 54]. The result also illustrates MAAIPM's superiority over symmetric Gauss-Seidel ADMM (sGS-ADMM) [53].

All experiments are run in Matlab R2018b on a workstation with two processors, Intel(R) Xeon(R) Processor E5-2630@2.40Ghz (8 cores and 16 threads per processor) and 64GB of RAM, equipped with 64-bit Windows 10 OS. Full experiment details are available in the appendix.

**Experiments on solving the normal equations:** For figure 2, one can see that both SLRM and DLRM clealy outperform the Matlab solver in in all cases. For computation time, SLRM increases linearly with respect to $N$ and $m'$, and DLRM increases linearly with respect to $N$ and $m$, which matches the conclusions in Theorem 4.3. In practice, we select SLRM when $m^2 \leq 4\sum_{t=1}^{N} m_t^2$ and DLRM when $m^2 > 4\sum_{t=1}^{N} m_t^2$.

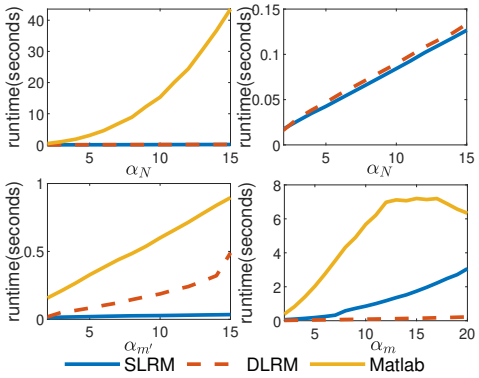



Figure 2: Average computation time of 200 independent trials in solving the linear system. Entries of diagonal $D$ and $\boldsymbol{f}$ are generated by uniform distribution in $(0, 1)$. In base situation, $N = 50, m = 50, m' = 25$. Sub-figures show the computation times when rescaling $N, m$ and $m_1 = \cdots = m_N = m'$ by respectively $\alpha_N$, $\alpha_m$ and $\alpha_{m'}$ times.

**Experiments on barycenter problems:** In this experiment, we set $d = 3$ for convenience. For $\mathcal{P}^{(t)}$, each entry of $(\boldsymbol{q}_1^{(t)}, \ldots, \boldsymbol{q}_{m'}^{(t)})$ is generated with i.i.d. standard Gaussian distribution. The entries of the weight vectors $(a_1^{(t)}, \ldots, a_{m'}^{(t)})$ are simulated by uniform distribution on $(0, 1)$ and then are normalized. Next we apply the $k$-means[2] method to choose $m$ points to be the support points. Note that Gurobi and MOSEK use a crossover strategy when close to the exact solution to ensure obtaining a highly accurate solution, we can regard Gurobi's objective value $\mathcal{F}_{gu}$ as the exact optimal value of the linear program (4). Let "normalized obj" denote the normalized objective value defined by $|\mathcal{F}_{method} - \mathcal{F}_{gu}|/\mathcal{F}_{gu}$, where $\mathcal{F}_{method}$ is the objective value respectively obtained by each method. Let "feasibility error" denote

[2]We call the Malab function "kmeans" in statistics and machine learning toolbox.

$\max\left\{\frac{\|\{\Pi^{(t)}\mathbf{1}_{m_t}-\boldsymbol{w}\}\|_F}{1+\|\boldsymbol{w}\|_F+\|\{\Pi^{(t)}\}\|_F}, \frac{\|\{(\Pi^{(t)})^\top\mathbf{1}_m-\boldsymbol{a}^{(t)}\}\|_F}{1+\|\{\boldsymbol{a}^{(t)}\}\|_F+\|\{\Pi^{(t)}\}\|_F}, |\mathbf{1}^\top\boldsymbol{w}-1|\right\}$, as a measure of the distance to the feasible set.

From figure 3, we see that MAAIPM displays a super-linear convergence rate for the objective, which is consistent with the result of [56]. Note that the feasibility error of MAAIPM increases a little bit near the end but is still much lower than BADMM and IBP. Although other methods may have lower objective values in early stages, their solutions are not acceptable due to high feasibility errors.

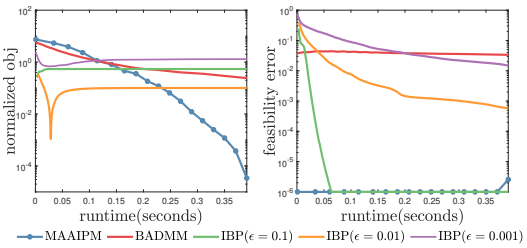



Figure 3: Performance of methods in pre-specified support cases. $N = m = 50$ and $m_1 = \cdots = m_N = 50$

Then we run numerical experiments to test the computation time of methods in pre-specified support points cases. For MAAIPM, we terminate it when $(\boldsymbol{b}^\top\boldsymbol{\lambda}_k - \boldsymbol{c}^\top\boldsymbol{x}_k)/(1+|\boldsymbol{b}^\top\boldsymbol{\lambda}_k| + |\boldsymbol{c}^\top\boldsymbol{x}_k|)$ is less than $5\times10^{-5}$. For sGS-ADMM, we compare with it indirectly by the benchmark claimed in their paper [53]: commercial solver Gurobi 8.1.0 [24] (academic license) with the default parameter settings. We also compare with another commercial solver MOSEK 9.1.0(academic license). In our observation, MAAIPM can frequently perform better than other popular commercial solvers. We use the default parameter setting(optimal for most cases) for Gurobi and MOSEK so that they can exploit multiple processors (16 threads) while other methods are implemented with only one thread[3]. For BADMM, we follow the algorithm 4 in [54] to implement and terminate when $\|\Pi^{(k,1)}-\Pi^{(k,2)}\|_F/(1+\|\Pi^{(k,1)}\|_F+\|\Pi^{(k,2)}\|_F) < 10^{-5}$. Set $\|\{A_t\}\|_F = \left(\sum_{t=1}^N\|A_t\|_F^2\right)^{\frac{1}{2}}$. For IBP, we follow the remark 3 in [9] to implement the method, terminate it when $\|\{u_k^{(n)}\}-\{u_k^{(n-1)}\}\|_F/(1+\|\{u_k^{(n)}\}\|_F+\|\{u_k^{(n-1)}\}\|_F) < 10^{-8}$ and $\|\{v_k^{(n)}\}-\{v_k^{(n-1)}\}\|_F/(1+\|\{v_k^{(n)}\}\|_F+\|\{v_k^{(n-1)}\}\|_F) < 10^{-8}$, and choose the regularization parameter $\epsilon$ from $\{0.1, 0.01, 0.001\}$ in our experiments. For BADMM and IBP, we implement the Matlab codes[4] by J.Ye et al. [54] and set the maximum iterate number respectively 4000 and $10^5$.

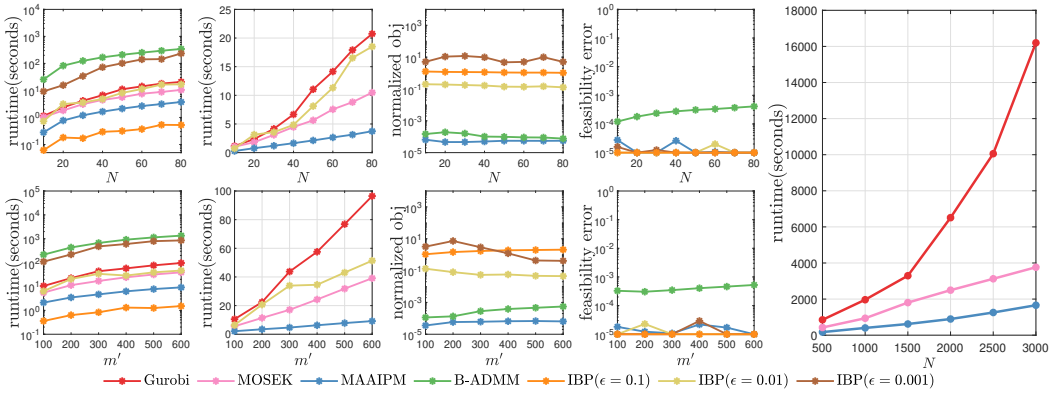



Figure 4: The left 8 figures are the average computation time, normalized objective value and feasibility error of Gurobi, MOSEK, MAAIPM, BADMM and IBP($\epsilon = 0.1,\ 0.01,\ 0.001$) in pre-specified support cases from 30 independent trials. In the first row, $m = 100$, $m_t$ follows an uniform distribution on $(75, 125)$. In the second row, $N = 50,\ m = 100$ and $m_1 = \cdots = m_N = m'$. The right figure is the average computation time of Gurobi and MAAIPM in pre-specified support cases from 10 independent trials. $m_t$ follows a uniform distribution on $(150, 250)$, and $m = 200$.

From the left 8 sub-figures in figure 4 one can observe that MAAIPM returns a considerably accurate solution in the second shortest computation time. For IBP, although it returns an objective value in the shortest time when $\epsilon = 0.1$, the quality of the solution is almost the worst. Because IBP only solves an approximate problem, if $\epsilon$ is set smaller, the computation time sharply increases but the

[3]We call the Matlab function "maxNumCompThreads(1)"

[4]Available in https://github.com/bobye/WBC_Matlab

quality of the solution is still not ensured. For BADMM, it gives a solution close to the exact one, but requires much more computation time.

For Gurobi and MOSEK, although they can exploit 16 threads, the computation time is far more than that of MAAIPM That is to say, MAAIPM also largely outperforms sGS-ADMM in speed, according to table 1, 2, 3 in [53]. Moreover, because the number of iterations remains almost independent of the problem size, the main computational cost of MAAIPM is approximately linear with respect to $N$ and $m'$. In fact, when $N = 5000$, MAAIPM requires only 3098.23 seconds, while MOSEK uses over 20000 seconds. Although the memory usage of MAAIPM is within a constant multiple of that of IBP, the former one is ususaly larger than the latter one. But the right subfigure in Figure 4 and the case of $N = 5000$ demonstrate that MAAIPM's memory usage is managed more efficient compared to Gurobi and MOSEK. These positive traits are consistent with the time and memory complexity proved in Theorem 4.3.

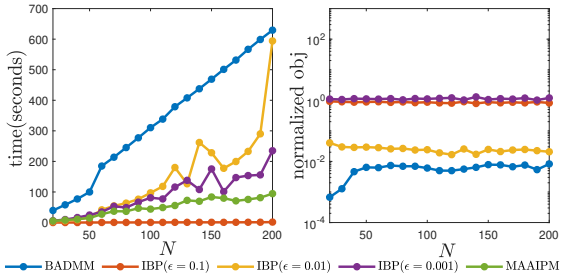



Figure 5: computation time and normalized objective value of MAAIPM, BADMM and IBP in the free support cases from 30 independent trials. "Normalized obj" denote $\mathcal{F}_{method}/\mathcal{F}_{MAAIPM} - 1$, where $\mathcal{F}_{method}$ is the objective value obtained by each method. $N$ takes different values and $m = m' = 50$.

Next, we conduct numerical studies to test MAAIPM in free support cases, i.e., problem (3). Same as [54], we implement the version of BADMM and IBP that can automatically update support points and set the initial support points in multivariate normal distribution. We set the maximum number of iterations in BADMM and IBP as $10^4$ and $10^6$. The entries of $(\boldsymbol{q}_1^{(t)}, \ldots, \boldsymbol{q}_{m'}^{(t)})$ are generated with i.i.d. uniform distribution in $(0, 100)$ and the initial support points follows a Gaussian distribution. In figure 6, "Normalized obj" denotes $\mathcal{F}_{method}/\mathcal{F}_{MAAIPM} - 1$, where $\mathcal{F}_{method}$ is the objective value obtained by each iteration of methods. From figure 5 and 6, one can see that, in the free support cases, MAAIPM can still obtain the smallest objective value in the second shortest time. That is because MAAIPM updates support more frequently and adopts "jump" tricks to avoid the local minima. Although IBP can obtain an approximate value in the shortest time when $\epsilon = 0.1$, the quality of the barycenter is too low to be useful.

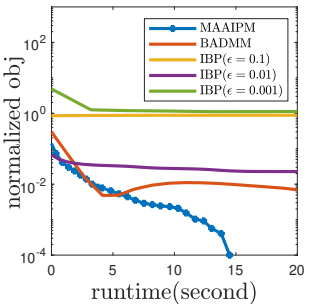
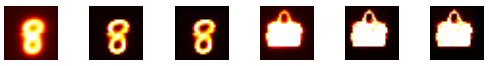
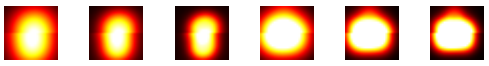
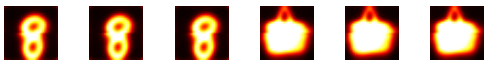



Figure 6: Performance of methods in free support cases. $N = 40$, $m = m_1 = m_2 = \cdots = m_N = 50$.

**Experiments on real applications:** We conduct similar experiments to [16, 53] on the MNIST[4] and Fashion-MNIST[4] datasets. In MNIST, We randomly select 200 images for digit 8 and resize each image to 0.5, 1, 2 times of its original size $28 \times 28$. In Fashion-MNIST, we randomly select 20 images of handbag, and resize each image to 0.5, 1 time of the original size. The support points of images are dense and different. Next, for each case, we apply MAAIPM, BADMM and IBP($\epsilon = 0.01$) to compute the Wasserstein barycenter in respectively free support cases and pre-specified support cases. From table 1, one can see that, MAAIPM obtained the clearest and sharpest barycenters within the least computation time.

Table 1: Experiments on datasets

| | MNIST | | | Fashion-MNIST | | |
|---|---|---|---|---|---|---|
| time(seconds) | 250 | 500 | 1000 | 25 | 50 | 75 |
| MAAIPM |  | | | | | |
| BADMM |  | | | | | |
| IBP($\epsilon = 0.01$) |  | | | | | |

[4] Available in http://yann.lecun.com/exdb/mnist/ and https://github.com/zalandoresearch/fashion-mnist

**Acknowledgments**

We thank Tianyi Lin, Simai He, Bo Jiang, Qi Deng and Yibo Zeng for helpful discussions and fruitful suggestions.

## References

[1] S.S. Abadeh, V.A. Nguyen, D. Kuhn, and P.M.M Esfahani. Wasserstein distributionally robust kalman filtering. In *Advances in Neural Information Processing Systems 31*, pages 8483–8492, 2018.

[2] M. Agueh and G. Carlier. Barycenters in the Wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2): 904–924, 2011.

[3] J. Altschuler, J. Weed, and P. Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Advances in Neural Information Processing Systems 30*, pages 1964–1974, 2017.

[4] P.C. Alvarez-Esteban, E. Barrio, J. Cuesta-Albertos, and C. Matran. A fixed-point approach to barycenters in Wasserstein space. *Journal of Mathematical Analysis and Applications*, 441(2): 744–762, 2016.

[5] S. Amari, R. Karakida, M.Oizumi and M. Cuturi Information geometry for regularized optimal transport and barycenters of patterns. *Neural computation*, 31(5): 827-848, 2019.

[6] E. Anderes, S. Borgwardt, and J. Miller. Discrete Wasserstein barycenters: Optimal transport for discrete data. *Math Meth Oper Res*,84(2):389–409,October2016. ISSN 1432-2994, 1432-5217. doi: 10.1007/s00186-016-0549-x.

[7] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[8] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[9] J.D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

[10] J. Blanchet, A. Jambulapati, C. Kent and A. Sidford. Towards optimal running times for optimal transport. *arXiv:1810.07717*.

[11] G. Carlier, A. Oberman, and E. Oudet. Numerical methods for matching for teams and Wasserstein barycenters. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1621–1642, 2015.

[12] S. Claici, E. Chien, J. Solomon. Stochastic Wasserstein Barycenters. *arXiv:1802.05757*.

[13] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *Advances in Neural Information Processing Systems 30*, pages 3730–3739. 2017.

[14] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26*, pages 2292–2300, 2013.

[15] A. Dessein, N. Papadakis, and C.A. Deledalle. Parameter estimation in finite mixture models by regularized optimal transport: a unified framework for hard and soft clustering. *arXiv:1711.04366*, 2017.

[16] M. Cuturi and A. Doucet. Fast computation of Wasserstein barycenters. In *the 31st International Conference on Machine Learning*, pages 685–693, 2014.

[17] P. Dvurechenskii, D. Dvinskikh, A. Gasnikov, C. Uribe, and A. Nedich. Decentralize and randomize: Faster algorithm for Wasserstein barycenters. In *Advances in Neural Information Processing Systems 30*, pages 10783–10793, 2018.

[18] P. Dvurechensky, A. Gasnikov and A. Kroshnin. Computational Optimal Transport: Complexity by Accelerated Gradient Descent Is Better Than by Sinkhorn's Algorithm. *Proceedings of the 35th International Conference on Machine Learning*, 80:1367-1376, 2018.

[19] C. Frogner, C. Zhang, H. Mobahi, M. Araya, and T.A. Poggio. Learning with a Wasserstein loss. In *Advances in Neural Information Processing Systems 28*, pages 2053–2061, 2015.

[20] F. L. Gall and F. Urrutia. Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor. *arXiv:1708.05622*. 2017

[21] R. Gao, L. Xie, Y. Xie, and H. Xu. Robust hypothesis testing using wasserstein uncertainty sets. In *Advances in Neural Information Processing Systems 31*, pages 7913–7923. 2018.

[22] A. Genevay, M. Cuturi, G. Peyré, and F. Bach. Stochastic optimization for large-scale optimal transport. In *Advances in Neural Information Processing Systems 29*, pages 3440–3448, 2016.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A.C. Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777, 2017.

[24] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 2018.

[25] N. Ho, V. Huynh, D. Phung, and M.I. Jordan. Probabilistic multilevel clustering via composite transportation distance. *arXiv:1810.11911*, 2018.

[26] N. Ho, X. Nguyen, M. Yurochkin, H.H. Bui, V. Huynh, and D. Phung. Multilevel clustering via wasserstein means. In *International Conference on Machine Learning*, pages 1501–1509, 2017.

[27] G. Huang, C. Guo, M.J. Kusner, Y. Sun, F. Sha, and K.Q. Weinberger. Supervised word mover's distance. In *Advances in Neural Information Processing Systems 29*, pages 4862– 4870, 2016.

[28] E. John, E. A. Yıldırım. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, 41(2): 151-183, 2008.

[29] E. A. Yıldırım, S. J. Wright Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3): 782-810, 2002.

[30] M.J. Kusner, Y. Sun, N.I. Kolkin, and K.Q. Weinberger. From word embeddings to document distances. In *the 32nd In International Conference on Machine Learning*, pages 957–966, 2015.

[31] T. Lacombe, M. Cuturi, and S. Oudot. Large scale computation of means and clusters for persistence diagrams using optimal transport. In *Advances in Neural Information Processing Systems 31*, pages 9792–9802, 2018.

[32] J. Lee and M. Raginsky. Minimax statistical learning with wasserstein distances. In *Advances in Neural Information Processing Systems 31*, pages 2692–2701. 2018.

[33] T. Lin, N. Ho and M.I. Jordan. On Efficient Optimal Transport: An Analysis of Greedy and Accelerated Mirror Descent Algorithms. *arXiv:1901.06482*.

[34] A. Mallasto and A. Feragen. Learning from uncertain curves: The 2-wasserstein metric for gaussian processes. In *Advances in Neural Information Processing Systems 30*, pages 5660–5670. 2017.

[35] S. Mehrotra. On the Implementation of a Primal-Dual interior-point Method. *SIAM J. Optim.*, 2(4), 575–601. 1992.

[36] S. Mizuno, M. J. Todd and Y. Ye. On adaptive-step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations research*, 18(4): 964-981, 1993.

[37] B. Muzellec and M. Cuturi. Generalizing point embeddings using the wasserstein space of elliptical distributions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa- Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10258–10269. 2018.

[38] X. Nguyen. Convergence of latent mixing measures in finite and infinite mixture models. *Annals of Statistics*, 4(1):370–400, 2013.

[39] X. Nguyen. Borrowing strength in hierarchical Bayes: posterior concentration of the Dirichlet base measure. *Bernoulli*, 22(3):1535–1571, 2016.

[40] J. Nocedal and S.J. Wright *Numerical Optimization*, 2006.

[41] G. Peyré, M. Cuturi, and J. Solomon. Gromov-Wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning*, pages 2664–2672, 2016.

[42] G. Peyré and M. Cuturi. Computational Optimal Transport. *arXiv:1803.00567*.

[43] J. Rabin, G. Peyré, J. Delon, and M. Bernot. Wasserstein barycenter and its application to texture mixing. In *Scale Space and Variational Methods in Computer Vision*, volume 6667 of *Lecture Notes in Computer Science*, pages 435–446. Springer, 2012.

[44] A. Rolet, M. Cuturi, and G. Peyré. Fast dictionary learning with a smoothed Wasserstein loss. In *International Conference on Artificial Intelligence and Statistics*, pages 630–638, 2016.

[45] A. Skajaa, E. D. Andersen, Y. Ye Warmstarting the homogeneous and self-dual interior-point method for linear and conic quadratic problems. *Mathematical Programming Computation*, 5(1): 1-25, 2013.

[46] J. Solomon, G. D. Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, D. Tao, L. Guibas, Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4), 66, 2015.

[47] J. Solomon, R.M. Rustamov, L. Guibas, and A. Butscher. Wasserstein propagation for semi-supervised learning. In *the 31st International Conference on Machine Learning*, pages 306–314, 2014.

[48] S. Srivastava, C. Li, and D. Dunson. Scalable Bayes via barycenter in Wasserstein space. *Journal of Machine Learning Research*, 19(8):1–35, 2018.

[49] M. Staib, S. Claici, J.M. Solomon, and S. Jegelka. Parallel streaming Wasserstein barycenters. In *Advances in Neural Information Processing Systems 30*, pages 2647–2658, 2017a.

[50] C. Villani. *Optimal Transport: Old and New*, volume 338. Springer Science & Business Media, 2008.

[51] H. Wang and A. Banerjee. Bregman Alternating Direction Method of Multipliers. In *Advances in Neural Information Processing Systems 27*, 2014, pp. 2816-2824.

[52] S.J. Wright. *Primal-dual interior-point methods*, 1997.

[53] L. Yang, J. Li, D. Sun and K.C. Toh. A Fast Globally Linearly Convergent Algorithm for the Computation of Wasserstein Barycenters, *arXiv:1809.04249*, 2018.

[54] J. Ye, P. Wu, J.Z. Wang, and J. Li. Fast discrete distribution clustering using Wasserstein barycenter with sparse support. *IEEE Transactions on Signal Processing*, 65(9):2317–2332, 2017.

[55] Y Ye. On affine scaling algorithms for nonconvex quadratic programming. *Mathematical Programming*, 56(1-3): 285-300, 1992

[56] Y. Ye, O. Güler, R. A. Tapia, and Y. Zhang. A quadratically convergent O ($\sqrt{n}L$)-iteration algorithm for linear programming. *Mathematical Programming* , 59(1):151-162 2014.

# A Proof of lemma 3.1

To justify the claims in lemma 3.1, we follows the following line of proof: a) First, we show that through a series of row transformations, we can transform matrix A into a matrix whose elements in $(M+1)$-th, $(M+m+1)$-th, $\cdots$, $(M+(N-1)m+1)$-th rows are zeros, and elements in other positions are the same as A. b) Second, we prove that the matrix $\bar{A}$ has full row rank.

a). From the definition of matrix A, we have

$$A = \begin{bmatrix} F_1 & & & & \\ & F_2 & & & \\ & & \ddots & & \\ & & & F_N & \\ G_1 & & & & -I_m \\ & G_2 & & & -I_m \\ & & \ddots & & \vdots \\ & & & G_N & -I_m \\ & & & & \mathbf{1}_m^T \end{bmatrix} \tag{13}$$

where $F_i = I_{m_i} \otimes \mathbf{1}_m^\top$, $G_i = \mathbf{1}_{m_i}^\top \otimes I_m$ for $i = 1, ..., N$.

Let

$$\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{m\times 1}, \quad T_i = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ & & & \end{bmatrix}_{m\times m_i}, \quad S_i = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{m\times m}, \ i = 1, ..., N$$

and

$$L_1 = \begin{bmatrix} I_{m_1} & & & & & & \\ & \ddots & & & & & \\ & & I_{m_N} & & & & \\ -T_1 & & & I_m & & & \\ & \ddots & & & \ddots & & \\ & & -T_N & & & I_m & \\ & & & & & & 1 \end{bmatrix}, \; L_2 = \begin{bmatrix} I_{m_1} & & & & & & \\ & \ddots & & & & & \\ & & I_{m_N} & & & & \\ & & & S_1 & & & \boldsymbol{e}_1 \\ & & & & \ddots & & \vdots \\ & & & & & S_N & \boldsymbol{e}_1 \\ & & & & & & 1 \end{bmatrix}$$

Then

$$L_2 L_1 A = \begin{bmatrix} F_1 & & & & & \\ & F_2 & & & & \\ & & \ddots & & & \\ & & & F_N & & \\ G_1^{(1)} & & & & H^{(1)} & \\ & G_2^{(1)} & & & H^{(1)} & \\ & & \ddots & & \vdots & \\ & & & G_N^{(1)} & H^{(1)} & \\ & & & & \mathbf{1}_m^T & \end{bmatrix},$$

where $G_i^{(1)} = S_i G_i - S_i T_i F_i$, $H^{(1)} = \boldsymbol{e}_1 \mathbf{1}_m^\top - S_i$. It is easy to verify that elements in the first rows of $H^{(1)}$ and $G_i^{(1)}, i = 1, ..., N$ are zeros. We have proved the claims in a).

b). As defined in the claims of lemma 3.1, $\bar{A}$ is obtained by removing the $(M+1)$-th, $(M+m+1)$-th, $\cdots$, $(M + (N-1)m + 1)$-th rows of A. That is,

$$\bar{A} = \begin{bmatrix} F_1 & & & & & \\ & F_2 & & & & \\ & & \ddots & & & \\ & & & F_N & & \\ G_1^{(2)} & & & & H^{(2)} & \\ & G_2^{(2)} & & & H^{(2)} & \\ & & \ddots & & \vdots & \\ & & & G_N^{(2)} & H^{(2)} & \\ & & & & \mathbf{1}_m^\top & \end{bmatrix}$$

where $G_i^{(2)} = G_i^{(1)}(2:m,:) = \mathbf{1}_{m_i}^\top \otimes [\mathbf{0}_{m-1}, I_{m-1}]$, $H^{(2)} = H^{(1)}(2:m,:) = [\mathbf{0}_{m-1}, -I_{m-1}]$ and $F_i = I_{m_i} \otimes \mathbf{1}_m^\top$. Let $n'_{row} = M + N(m-1) + 1$.

For $i = 1, ..., N$, let

$$U_i = I_{m_i} \otimes \begin{bmatrix} 1 & -1 & \cdots & -1 \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{m \times m}, \; U_{N+1} = \begin{bmatrix} 1 & -1 & \cdots & -1 \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{m \times m}, \; R_1 = \begin{bmatrix} U_1 & & & \\ & U_2 & & \\ & & \ddots & \\ & & & U_{N+1} \end{bmatrix}$$

then

$$\bar{A}R_1 = \begin{bmatrix} F_1^{(3)} & & & & \\ & F_2^{(3)} & & & \\ & & \ddots & & \\ & & & F_N^{(3)} & \\ G_1^{(3)} & & & & H^{(3)} \\ & G_2^{(3)} & & & H^{(3)} \\ & & \ddots & & \vdots \\ & & & G_N^{(3)} & H^{(3)} \\ & & & & \boldsymbol{\alpha}^\top \end{bmatrix}$$

where $F_i^{(3)} = F_i U_i = I_{m_i} \otimes [1, \mathbf{0}_{m-1}^\top]$, $G_i^{(3)} = G_i^{(2)} U_i = G_i^{(2)} = \mathbf{1}_{m_i}^\top \otimes [\mathbf{0}_{m-1}, I_{m-1}]$, $i = 1, ..., N$, $H^{(3)} = H^{(2)} U_{N+1} = H^{(2)} = [\mathbf{0}_{m-1}, -I_{m-1}]$ and $\boldsymbol{\alpha}^\top = \mathbf{1}_m^\top U_{N+1} = [1, \mathbf{0}_{m-1}^\top]$.

Let

$$\tilde{K} = \begin{bmatrix} 0 & & & \\ & -1 & & \\ & & \ddots & \\ & & & -1 \end{bmatrix}_{m \times m}, \; K_i = \begin{bmatrix} I_m & \tilde{K} & \cdots & \tilde{K} \\ & I_m & & \\ & & \ddots & \\ & & & I_m \end{bmatrix}_{mm_i \times mm_i}, \; R_2 = \begin{bmatrix} K_1 & & & \\ & \ddots & & \\ & & K_N & \\ & & & I_m \end{bmatrix}$$

then

$$\bar{A}R_1R_2 = \begin{bmatrix} F_1^{(4)} & & & & \\ & F_2^{(4)} & & & \\ & & \ddots & & \\ & & & F_N^{(4)} & \\ G_1^{(4)} & & & & H^{(3)} \\ & G_2^{(4)} & & & H^{(3)} \\ & & \ddots & & \vdots \\ & & & G_N^{(4)} & H^{(3)} \\ & & & & \boldsymbol{\alpha}^\top \end{bmatrix}$$

where $F_i^{(4)} = F_i^{(3)} K_i = F_i^{(3)} = I_{m_i} \otimes [1, \mathbf{0}_{m-1}^\top]$, $G_i^{(4)} = G_i^{(3)} K_i = [\mathbf{0}_{m-1}, I_{m-1}, \mathbf{0}_{(m-1)\times(mm_i - m)}]$, $i = 1, ..., N$,

Let $\tilde{A}$ be the matrix composing of the first $(mM + 1)$ columns of $\bar{A}R_1R_2$. That is,

$$\tilde{A} = \begin{bmatrix} F_1^{(4)} & & & & \\ & F_2^{(4)} & & & \\ & & \ddots & & \\ & & & F_N^{(4)} & \\ G_1^{(4)} & & & & \\ & G_2^{(4)} & & & \\ & & \ddots & & \\ & & & G_N^{(4)} & \\ & & & & 1 \end{bmatrix}$$

Matrix $\tilde{A}$ satisfies two properties:

(1) Each row of $\tilde{A}$ has one and only one nonzero element (being 1) with other elements being 0;

(2) Each column of $\tilde{A}$ has at most one nonzero element.

Therefore, there exists permutation matrices $P_1 \in \mathbb{R}^{n'_{row}}$ and $Q_1 \in \mathbb{R}^{n_{col}-m+1}$ such that $P_1\tilde{A}Q_1 = [I_{n'_{row}}, 0_{n'_{row}\times(Mm+1)}]$. Thus $rank(\tilde{A}) = rank(P_1\tilde{A}Q_1) = n'_{row}$ and $rank(\bar{A}) = n'_{row}$.

## B Proof of lemma 4.1

In this subsection, we give the proof of lemma 4.1.

*Proof.* Let $\boldsymbol{d}$ be the diagonal vector of matrix $D$; $M := \sum_{i=1}^{N} m_i$ and $M_2 := N(m-1)$. Same as the preceding section, the structure of $\bar{A}$ as:

$$\bar{A} = \begin{bmatrix} F_1 & & & & \\ & F_2 & & & \\ & & \ddots & & \\ & & & F_N & \\ G_1^{(2)} & & & & H^{(2)} \\ & G_2^{(2)} & & & H^{(2)} \\ & & \ddots & & \vdots \\ & & & G_N^{(2)} & H^{(2)} \\ & & & & \mathbf{1}_m^\top \end{bmatrix}$$

where $G_i^{(2)} = G_i^{(1)}(2:m,:) = \mathbf{1}_{m_i}^\top \otimes [\mathbf{0}_{m-1}, I_{m-1}]$, $H^{(2)} = H^{(1)}(2:m,:) = [\mathbf{0}_{m-1}, -I_{m-1}]$ and $F_i = I_{m_i} \otimes \mathbf{1}_m^\top$.

Let

$$\bar{A}_1 := \bar{A}(1:M,:) = \begin{bmatrix} F_1 & & & \\ & F_2 & & \\ & & \ddots & \\ & & & F_N \end{bmatrix},$$

$$\bar{A}_2 := \bar{A}(M+1:M+(m-1)N,:) = \begin{bmatrix} G_1^{(2)} & & & & H^{(2)} \\ & G_2^{(2)} & & & H^{(2)} \\ & & \ddots & & \vdots \\ & & & G_N^{(2)} & H^{(2)} \end{bmatrix},$$

$$\bar{A}_3 := \bar{A}(M+(m-1)N+1,:) = \begin{bmatrix} & & & \mathbf{1}_m^\top \end{bmatrix}.$$

Then

$$\bar{A} = \begin{bmatrix} \bar{A}_1 \\ \bar{A}_2 \\ \bar{A}_3 \end{bmatrix} \text{ and } \bar{A}D\bar{A}^\top = \begin{bmatrix} \bar{A}_1 D\bar{A}_1^\top & \bar{A}_1 D\bar{A}_2^\top & \bar{A}_1 D\bar{A}_3^\top \\ \bar{A}_2 D\bar{A}_1^\top & \bar{A}_2 D\bar{A}_2^\top & \bar{A}_2 D\bar{A}_3^\top \\ \bar{A}_3 D\bar{A}_1^\top & \bar{A}_3 D\bar{A}_2^\top & \bar{A}_3 D\bar{A}_3^\top \end{bmatrix}.$$

Now we analyze the structure of each sub-matrix $\bar{A}_i D\bar{A}_j^\top$ and rename them for conciseness. Let

$$D = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_{N+1} \end{bmatrix},$$

where $D_i \in \mathbb{R}^{mm_i \times mm_i}$, $i = 1, \ldots, N$ and $D_{N+1} \in \mathbb{R}^{m\times m}$. Then

$$\bar{A}_1 D\bar{A}_1^\top = \begin{bmatrix} F_1 D_1 F_1^\top & & \\ & \ddots & \\ & & F_N D_N F_N^\top \end{bmatrix} := B_1.$$

Each $F_i D_i F_i^\top$ is a diagonal matrix with positive diagonal entries.

$$\bar{A}_2 D \bar{A}_1^\top = \begin{bmatrix} G_1^{(2)} D_1 F_1^\top & & \\ & \ddots & \\ & & G_N^{(2)} D_N F_N^\top \end{bmatrix} := B_2^\top,$$

$$\bar{A}_2 D \bar{A}_2^\top = \begin{bmatrix} G_1^{(2)} D_1 G_1^{(2)\top} & & \\ & \ddots & \\ & & G_N^{(2)} D_N G_N^{(2)\top} \end{bmatrix} + \begin{bmatrix} H^{(2)} D_{N+1} H^{(2)\top} & \cdots & H^{(2)} D_{N+1} H^{(2)\top} \\ \vdots & & \vdots \\ H^{(2)} D_{N+1} H^{(2)\top} & \cdots & H^{(2)} D_{N+1} H^{(2)\top} \end{bmatrix}. \tag{14}$$

where $H^{(2)} D_{N+1} H^{(2)\top}$ and each $G_i^{(2)} D_i G_i^{(2)\top}$ is a diagonal matrix with positive diagonal entries. We use $B_3$ to denote the first matrix in the right hand side of (14) and $B_4$ to denote the second. In addition, other blocks of $\bar{A} D \bar{A}^\top$ are

$$\bar{A}_3 D \bar{A}_1^\top = 0,$$

$$\bar{A}_3 D \bar{A}_2^\top = \begin{bmatrix} \mathbf{1}_m^\top D_{N+1} H^{(2)\top} & \cdots & \mathbf{1}_m^\top D_{N+1} H^{(2)\top} \end{bmatrix} := \boldsymbol{\alpha}^\top,$$

$$\bar{A}_3 D \bar{A}_3^\top = \mathbf{1}_m^\top D_{N+1} \mathbf{1}_m := c.$$

With the new notations, we have

$$\bar{A} D \bar{A}^T = \begin{bmatrix} B_1 & B_2 & \mathbf{0} \\ B_2^\top & B_3 + B_4 & \boldsymbol{\alpha} \\ \mathbf{0} & \boldsymbol{\alpha}^\top & c \end{bmatrix}.$$

□

# C Proof of lemma 4.2

To justify lemma 4.2, we need the following basic result which can be verified through direct computation.

**Lemma C.1** *All the non-zero entries of matrices $B_1, B_2, B_3$ and $B_4$ are positive, and*

***a)*** $B_3 \mathbf{1}_{M_2} = B_2^\top \mathbf{1}_M$. ***b)*** $B_1 \mathbf{1}_M - B_2 \mathbf{1}_{M_2} > 0$.

*Proof.* a)

$$B_3 \mathbf{1}_{M_2} = \begin{bmatrix} G_1^{(2)} D_1 G_1^{(2)\top} \mathbf{1}_{m-1} \\ \vdots \\ G_N^{(2)} D_N G_N^{(2)\top} \mathbf{1}_{m-1} \end{bmatrix}, \quad B_2^\top \mathbf{1}_M = \begin{bmatrix} G_1^{(2)} D_1 F_1^\top \mathbf{1}_{m_1} \\ \vdots \\ G_N^{(2)} D_N F_N^\top \mathbf{1}_{m_N} \end{bmatrix}$$

Recall that $G_i^{(2)} = \mathbf{1}_{m_i}^\top \otimes [\mathbf{0}_{m-1}, I_{m-1}]$, $F_i = I_{m_i} \otimes \mathbf{1}_m^\top$, and $D_i$'s are diagonal matrices, we have $G_i^{(2)} D_i F_i^\top \mathbf{1}_{m_i} = G_i^{(2)} D_i G_i^{(2)\top} \mathbf{1}_{m-1}$ and thus $B_3 \mathbf{1}_{M_2} = B_2^\top \mathbf{1}_M$.

b)

$$B_1 \mathbf{1}_M = \begin{bmatrix} F_1 D_1 F_1^\top \mathbf{1}_{m_1} \\ \vdots \\ F_N D_N F_N^\top \mathbf{1}_{m_N} \end{bmatrix}, \quad B_2 \mathbf{1}_{M_2} = \begin{bmatrix} F_1 D_1 G_1^{(2)\top} \mathbf{1}_{m-1} \\ \vdots \\ F_N D_N G_N^{(2)\top} \mathbf{1}_{m-1} \end{bmatrix}$$

It is easy to verify that $F_i D_i F_i^\top \mathbf{1}_{m_i} > F_i D_i G_i^{(2)\top} \mathbf{1}_{m-1}$ and thus $B_1 \mathbf{1}_M - B_2 \mathbf{1}_{M_2} > 0$. □

With this basic lemma at hand, we are able to prove lemma 4.2.

**proof of lemma 4.2:**

*Proof.*

**a)** It is easy to verify the block-diagonal structure of $A_1$, so we just need to prove the positive definiteness and the strict diagonal dominance. Assume $A_1$ is not positive definite and $-\lambda \leq 0$ is an eigenvalue of $A_1$, then $\lambda I_{M_2} + A_1$ is a singular matrix.

From the results in lemma C.1, we have

$$\begin{aligned} & (\lambda I_{M_2} + A_1)\mathbf{1}_{M_2} \\ = \; & \lambda \mathbf{1}_{M_2} + B_3 \mathbf{1}_{M_2} - (B_2^\top B_1^{-1} B_2)\mathbf{1}_{M_2} \\ = \; & \lambda \mathbf{1}_{M_2} + B_2^\top B_1^{-1} B_1 \mathbf{1}_M - (B_2^\top B_1^{-1} B_2)\mathbf{1}_{M_2} \\ = \; & \lambda \mathbf{1}_{M_2} + B_2^\top B_1^{-1}(B_1 \mathbf{1}_M - B_2 \mathbf{1}_{M_2}) \\ > \; & \mathbf{0}_{M_2} \end{aligned} \tag{15}$$

where the first equality is from a) of lemma C.1; the last inequality is from b) of lemma C.1 and the fact that $B_2^\top B_1^{-1} \geq 0$ and each row of $B_2^\top B_1^{-1}$ has at least one strict positive entry.

Since $B_1, B_2, B_3 \geq 0$, $B_3$ is a diagonal matrix, together with (15), we know that the diagonal entries of $\lambda I_{M_2} + A_1 = \lambda I_{M_2} + B_3 - B_2^\top B_1^{-1} B_2$ are positive and the off-diagonal entries are non-positive. Let $E_{M_2} := \mathbf{1}_{M_2}\mathbf{1}_{M_2}^\top - I_{M_2}$, then

$$I_{M_2} \circ |A_1 + \lambda I_{M_2}| = I_{M_2} \circ (A_1 + \lambda I_{M_2}), \;\; E_{M_2} \circ |A_1 + \lambda I_{M_2}| = -E_{M_2} \circ (A_1 + \lambda I_{M_2}),$$

and

$$(I_{M_2} \circ |A_1 + \lambda I_{M_2}|)\, \mathbf{1}_{M_2} - (E_{M_2} \circ |A_1 + \lambda I_{M_2}|)\, \mathbf{1}_{M_2} = (\lambda I_{M_2} + A_1)\mathbf{1}_{M_2} > \mathbf{0}_{M_2}$$

This means $\lambda I_{M_2} + A_1$ is strictly diagonal dominant and thus nonsingular, which is a contradiction. Therefore, $A_1$ is positive definite. Take $\lambda = 0$ in the preceding analysis, we know $A_1$ is strictly diagonal dominant.

**b)** It is easy to verify that $A_2 = (\mathbf{1}_N \mathbf{1}_N^\top) \otimes (diag(\boldsymbol{y}) - \frac{1}{c}\boldsymbol{y}\boldsymbol{y}^\top)$. In view of the definition of $c$, we have $c > \mathbf{1}_{m-1}^\top \boldsymbol{y}$. Thus, the second claim of b) is a special case of a) with $B_1 = c$, $B_2 = \boldsymbol{y}^\top$ and $B_3 = diag(\boldsymbol{y})$.

□

## D Analysis of algorithm 2

In this section, we prove that through the steps in Algorithm 2, we get the accurate solution of the system $(A_1 + A_2)\boldsymbol{x} = \boldsymbol{g}$. We need a basic lemma on the inverse matrix on the sum of tow matrices.

**Lemma D.1** *Let $A \in \mathbb{R}^{n\times n}$ be an nonsingular matrix and $B \in \mathbb{R}^{n\times d}$, where $n$ and $d$ are two positive integers. Then*

$$(A + BB^\top)^{-1} = A^{-1} - A^{-1}B(I_n + B^\top A^{-1} B)^{-1} B^T A^{-1}$$

Recall that we have proved in lemma 4.2 that $Y$ is positive definite. Suppose $Y = R^\top R, R \in \mathbb{R}^{(m-1)\times(m-1)}$ and let $\tilde{R} = \mathbf{1}_N \otimes R^\top$. Then,$A_2 = \tilde{R}\tilde{R}^\top$. Further more, let

$$\bar{R} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}_{N\times 1} \otimes R^\top, \; and \; U = \begin{bmatrix} I_{m-1} & & & I_{m-1} \\ & I_{m-1} & & \vdots \\ & & \ddots & I_{m-1} \\ & & & I_{m-1} \end{bmatrix}_{M_2 \times M_2}$$

Note that $U$ is the same as defined in the main text part above Algorithm 2. It is easy to verify that $\tilde{R} = U\bar{R}$ and with the help of lemma D.1, we have

$$\begin{aligned}(A_1 + A_2)^{-1} &= \left(A_1 + \tilde{R}\tilde{R}^\top\right)^{-1} \\ &= A_1^{-1} - A_1^{-1}\tilde{R}(I + \tilde{R}^\top A_1^{-1}\tilde{R})^{-1}\tilde{R}^\top A_1^{-1} \\ &= A_1^{-1} - A_1^{-1}U\bar{R}(I + \tilde{R}^\top A_1^{-1}\tilde{R})^{-1}\bar{R}^\top U^\top A_1^{-1}.\end{aligned}$$

Define

$$W := \bar{R}(I + \tilde{R}^\top A_1^{-1}\tilde{R})^{-1}\bar{R}^\top = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ R^\top \end{bmatrix} (I_{m-1} + \sum_{i=1}^{N} RA_{ii}R^\top)^{-1} \begin{bmatrix} 0 & \cdots & 0 & R \end{bmatrix} \tag{16}$$

then

$$(A_1 + A_2)^{-1} = A_1^{-1} - A_1^{-1}UWU^\top A_1^{-1} \tag{17}$$

From (16), it is clear that all entries of $W$ are zero, except for the last $(m-1) \times (m-1)$ block $W_{NN} = R^\top(I_{m-1} + \sum_{i=1}^{N} RA_{ii}R^\top)^{-1}R$. With further calculation,

$$W_{NN} = \left(R^{-1}R^{-\top} + \sum_{i=1}^{N} A_{ii}^{-1}\right)^{-1} = \left(Y^{-1} + \sum_{i=1}^{N} A_{ii}^{-1}\right)^{-1}.$$

To solve the system $(A_1 + A_2)\boldsymbol{x} = \boldsymbol{g}$ with the equation (17), we just need to let each term in (17) act on the vector step by step. That's exactly what Algorithm 2 does.

# E Proof of theorem 4.3

In this section, we present the detailed analysis of computational cost and memory usage of SLRM and DLRM. Here we restate theorem 4.3.

**Theorem E.1** *1). For SLRM, Algorithm 1, the time complexity in terms of flops is $O(m^2\sum_{i=1}^{N} m_i + Nm^3)$, and the memory usage in terms of doubles is $O(m\sum_{i=1}^{N} m_i + Nm^2)$. 2). For the DLRM, the time complexity in terms of flops is $O(m\sum_{i=1}^{N} m_i^2 + \sum_{i=1}^{N} m_i^3)$, and the memory usage in terms of doubles is $O(m\sum_{i=1}^{N} m_i + \sum_{i=1}^{N} m_i^2)$.*

*Proof.* (1) First, for SLRM, assuming taking full advantage of the sparse structure, we count the flops required for computing each of the following quantities in Algorithm 1:

$$B_1 : O(m\sum_{t=1}^{N} m_t);\ B_2 : 0;\ B_3 : O(m\sum_{t=1}^{N} m_t);\ T : O(m\sum_{t=1}^{N} m_t);\ A_1 : O(m^2\sum_{t=1}^{N} m_t);\ A_2 : O(m^2);$$

$$\boldsymbol{z}^{(1)} : O(m\sum_{t=1}^{N} m_t);\ \boldsymbol{z}^{(2)} : O(Nm);\ \boldsymbol{z}^{(3)} : O(Nm^3);\ \boldsymbol{z}^{(4)} : O(Nm);\ \boldsymbol{z}^{(5)} : O(m\sum_{t=1}^{N} m_t).$$

The computation of $A_1$ and $\boldsymbol{z}^{(3)}$ requires most flops. The total flops required for SLRM is $O(m^2\sum_{t=1}^{N} m_t + Nm^3)$.

On the other hand, for implementation of the whole interior-point methods, the major data that should be kept in the memory include:

(a) Several vectors that is at the same level as a primal variable or a dual variable. Note that the scale of a primal variable is $m(\sum_{i=1}^{N} m_i)+m$ flops, and the scale of a dual variable is $\sum_{i=1}^{N} m_i+N(m-1)+1$ flops.

(b) Matrix $\bar{A}$ which is defined in lemma 3.1. Recall that

$$\bar{A} = \begin{bmatrix} F_1 & & & & & \\ & F_2 & & & & \\ & & \ddots & & & \\ & & & F_N & & \\ G_1^{(2)} & & & & H^{(2)} \\ & G_2^{(2)} & & & H^{(2)} \\ & & \ddots & & \vdots \\ & & & G_N^{(2)} & H^{(2)} \\ & & & & \mathbf{1}_m^\top \end{bmatrix}$$

Since each column of $F_i$ and each column of $G^{(i)}$ has at most one non-zero element, the total number of non-zero elements in $F_1, \ldots, F_N$ and $G_1^{(2)}, \ldots, G_N^{(2)}$ is bounded by $2m\sum_{i=1}^N m_i$. In addition, $H^{(2)}$ has $m-1$ non-zero elements, so the total number of non-zero elements in $\bar{A}$ is bounded by $2m\sum_{i=1}^N m_i + N(m-1) + m$

(c) Diagonals of matrices $B_1$ and $B_3$, and diagonal blocks of matrices $B_2$ and $A_1$. The data scale of the diagonals of matrices $B_1$ and $B_3$ are even smaller than a dual variable. The diagonal blocks of matrices $B_2$ and $A_1$ have $m\sum_{i=1}^N m_i$ elements and $N(m-1)^2$ elements, respectively.

(d) Other intermediate vectors or matrices, whose data scale is bounded by a constant time of the data scale in (a), (b) and (c).

With the analysis in (a) to (d), we know the memory usage of SLRM is bounded by $O(m\sum_{i=1}^N m_i + Nm^2)$.

(2) The major difference of DLRM and SLRM is that, we don't need to formulate the diagonal blocks $\{A_{ii} : i = 1 \ldots, N\}$ of matrix $A_1$ explicitly and compute the inverses of $A_{ii}s$. Instead, we need to compute $(B_{1i} - B_{2i}B_{3i}^{-1}B_{2i}^\top)^{-1}$ explicitly, which requires $O(m\sum_{i=1}^N m_i^2)$ flops for matrix multiplication and $O(\sum_{i=1}^N m_i^3)$ flops for matrix inverse. Since all other matrix-vector operations are cheap compared with matrix multiplication and inverse, as a result, the leading cost of the computation time is at the level $O(m\sum_{i=1}^N m_i^2 + \sum_{i=1}^N m_i^3)$.

Further more, since we need to keep $(B_{1i} - B_{2i}B_{3i}^{-1}B_{2i}^\top)^{-1}, i = 1, \ldots, N$ in memory instead of $A_{ii}^{-1}$, with simply different analysis as in part(1), we know the memory usage of DLRM is at the level $O(m\sum_{i=1}^N m_i + \sum_{i=1}^N m_i^2)$.

□

# F Examples of local minima and saddle points in free support cases

**An example of local minima**

Set $\Pi^{(t)} = \left[{\boldsymbol{\pi}_1^{(t)}}^\top, {\boldsymbol{\pi}_2^{(t)}}^\top, \ldots, {\boldsymbol{\pi}_m^{(t)}}^\top\right]^\top$. Let $N$ be any positive integer and $m = 2$, $d = 1$, $m_t = 3$, $Q^{(t)} = [0, 0.9, 1.1]$ and $\boldsymbol{a}^t = [0.01, 0.495, 0.495]$. Then $X = [0, 1]$, $\boldsymbol{w} = (0.01, 0.99)$ and $\boldsymbol{\pi}_1^{(t)} = (0.01, 0, 0)$ and $\boldsymbol{\pi}_2^{(t)} = (0, 0.495, 0.495)$ is a local minimum. But it is not a global minimum because a lower objective value occurs when $X = \{0.9, 1.1\}$, $\boldsymbol{w} = (0.505, 0.495)$, $\boldsymbol{\pi}_1^{(t)} = (0.01, 0.495, 0)$ and $\boldsymbol{\pi}_2^{(t)} = (0, 0, 0.495)$.

**An example of saddle point**

Let $N$ be any positive integer and $m = 2$, $d = 1$, $m_t = 3$, $Q^{(t)} = [0, 1/2, 3/2]$ and $\boldsymbol{a}^t = [1/3, 1/3, 1/3]$, then $X = [0, 1]$, $\boldsymbol{w} = (1/3, 2/3)$, $\boldsymbol{\pi}_1^{(t)} = (1/3, 0, 0)$ and $\boldsymbol{\pi}_2^{(t)} = (0, 1/3, 1/3)$ is a saddle point. Fixing $X$, the $\boldsymbol{w}$ and $\Pi^{(t)}$ is an optimal basic solution of problem 4. Fixing $\boldsymbol{w}$ and $\Pi^{(t)}$, $X$ is the solution of (7). It is not a local minimum, because a lower objective value of problem 4 can occur when $X = \{\delta, 1\}, \forall\ \delta \in (0, 1/2)$.

## G Details of MAAIPM

Figure 7 visualizes the primal variables $x_i$ and objective gradients $c_i$ in each iteration of MAAIPM.

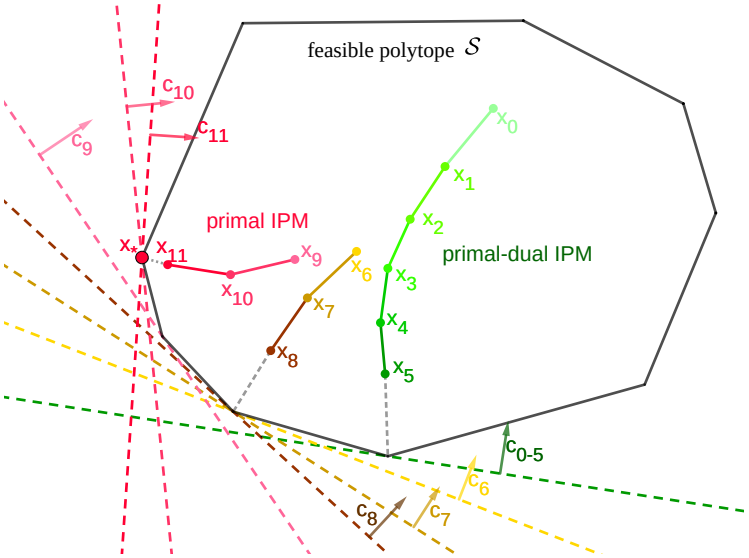



Figure 7: The primal variables and objective gradients in different iterations of MAAIPM. $x_i$ is returned by each iteration of IPM under the objective gradient $c_i$, and $c_i$, $i = 5, \ldots, 11$, is calculated by $x_{i-1}$ according to (8). At the beginning, MAAIPM updates objective gradient after every a few primal-dual IPM iterations(green). Then MAAIPM applies primal IPM(yellow and red) to frequently update objective gradient $c$ and uses "jump" tricks to escape local minima. $x_6$ and $x_9$ are the first primal variables returned by one primal IPM iteration form a smartly chosen starting point.

**Algorithm 3:** Matrix-based Adaptive Alternating Interior-point Method(MAAIPM)

**Input:** an initial $X^0$

**if** *support points are pre-specfied* **then**
    implement predictor-corrector IPM;
    **Output** $\boldsymbol{w}^*, \{\Pi^{(t),*}\}$

▷ Pre-specified support cases

**while** *at the beginning* **do**
    predictor-corrector IPM to solve (4) and update $X^*$;

▷ Update support $X^*$ every a few IPM iterations

**while** *a termination criterion is not met* **do**
    s = 0, apply the warm-start strategy to smartly choose the starting point;
    **while** *the penalty $\mu^s$ is not sufficiently close to* 0 **do**
        calculate the Newton direction $p^s$ at $(\boldsymbol{w}^s, \{\Pi^{(t),s}\})$ by (11);
        $(\boldsymbol{w}^{s+1}, \{\Pi^{(t),s+1}\}) = (\boldsymbol{w}^s, \{\Pi^{(t),s}\}) + \alpha^s p^s$, where $\alpha^s$ ensures the interior point;
        update $X^*$ by (7) and choose penalty $\mu^{s+1} < \mu^s$;

▷ Update support $X^*$ every IPM iteration

        $s = s + 1$;

▷ "Jump" tricks

**Output:** $\boldsymbol{w}^s, X^*, \{\Pi^{(t),s}\}$