Journal of Machine Learning Research 22 (2021) 1-37 Submitted 7/19; Revised 9/20; Published 1/21

A Fast Globally Linearly Convergent Algorithm for the Computation of Wasserstein Barycenters

Lei Yang MATYLEI@NUS.EDU.SG
Department of Mathematics
National University of Singapore
10 Lower Kent Ridge Road, Singapore 119076

Jia Li JIALI@STAT.PSU.EDU
Department of Statistics
Pennsylvania State University
University Park, PA 16802, USA

Defeng Sun DEFENG.SUN@POLYU.EDU.HK
Department of Applied Mathematics
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Kim-Chuan Toh MATTOHKC@NUS.EDU.SG
Department of Mathematics and Institute of Operations Research and Analytics
National University of Singapore
10 Lower Kent Ridge Road, Singapore 119076

Editor: Inderjit Dhillon

Abstract

We consider the problem of computing a Wasserstein barycenter for a set of discrete probability distributions with finite supports, which finds many applications in areas such as statistics, machine learning and image processing. When the support points of the barycenter are pre-specified, this problem can be modeled as a linear programming (LP) problem whose size can be extremely large. To handle this large-scale LP, we analyse the structure of its dual problem, which is conceivably more tractable and can be reformulated as a well-structured convex problem with 3 kinds of block variables and a coupling linear equality constraint. We then adapt a symmetric Gauss-Seidel based alternating direction method of multipliers (sGS-ADMM) to solve the resulting dual problem and establish its global convergence and global linear convergence rate. As a critical component for efficient computation, we also show how all the subproblems involved can be solved exactly and efficiently. This makes our method suitable for computing a Wasserstein barycenter on a large-scale data set, without introducing an entropy regularization term as is commonly practiced. In addition, our sGS-ADMM can be used as a subroutine in an alternating minimization method to compute a barycenter when its support points are not pre-specified. Numerical results on synthetic data sets and image data sets demonstrate that our method is highly competitive for solving large-scale Wasserstein barycenter problems, in comparison to two existing representative methods and the commercial software Gurobi.

Keywords: Wasserstein barycenter, discrete probability distribution, semi-proximal ADMM, symmetric Gauss-Seidel

1. Introduction

In this paper, we consider the problem of computing the mean of a set of discrete probability distributions under the Wasserstein distance (also known as the optimal transport distance *or* the earth mover's distance). This mean, called the Wasserstein barycenter, is also a discrete probability distribution (Agueh and Carlier, 2011). Recently, the Wasserstein barycenter has attracted much attention due to its promising performance in many application areas such as data analysis and statistics (Bigot and Klein, 2018), machine learning (Cuturi and Doucet, 2014; Li and Wang, 2008; Ye and Li, 2014; Ye et al., 2017) and image processing (Rabin et al., 2011). For a set of discrete probability distributions with finite support points, a Wasserstein barycenter with its support points being pre-specified can be computed by solving a linear programming (LP) problem (Anderes et al., 2016). However, the problem size can be extremely large when the number of discrete distributions *or* the number of support points of each distribution is large. Thus, classical LP methods such as the simplex method and the interior point method are no longer efficient enough or consume too much memory when solving this problem. This motivates the study of fast algorithms for the computation of Wasserstein barycenters; see, for example, (Benamou et al., 2015; Borgwardt, 2020; Borgwardt and Patterson, 2020; Carlier et al., 2015; Claici et al., 2018; Cuturi and Doucet, 2014; Cuturi and Peyré, 2016; Oberman and Ruan, 2015; Schmitzer, 2019; Solomon et al., 2015; Uribe et al., 2018; Xie et al., 2020; Ye and Li, 2014; Ye et al., 2017).

One representative approach is to introduce an entropy regularization in the LP and then apply some efficient first-order methods, e.g., the gradient descent method (Cuturi and Doucet, 2014) and the iterative Bregman projection (IBP) method (Benamou et al., 2015), to solve the regularized problem. These methods can be implemented efficiently and hence are suitable for large-scale data sets. However, they can only return an approximate solution of the LP (due to the entropy regularization) and often suffer from numerical instabilities and very slow convergence speed when the regularization parameter becomes small. The numerical issue can be alleviated by performing some stabilization techniques (e.g., the *log-sum-exp* technique) at the expense of losing some computational efficiency, but the slow speed may not be avoided. Thus, IBP is highly efficient if a rough approximate solution is adequate, as is the case in many learning tasks. However, our aim here is to obtain a high precision solution efficiently. Detailed empirical studies on the pros and cons of IBP are provided by Ye et al. (2017), specifically, in the scenario when the regularization parameter is reduced to obtain higher precision solutions. It was found that numerical difficulties often occur and the computational efficiency is lost when driving the regularization parameter to smaller values for obtaining more accurate solutions. We will also provide a comparison with IBP in our experiments. Another approach is to consider the LP as a constrained convex optimization problem with a separable structure and then apply some splitting methods to solve it. For example, the alternating direction method of multipliers (ADMM) was adapted in (Ye and Li, 2014). However, solving the quadratic programming subproblems involved is still highly expensive. Later, Ye et al. (2017) developed a modified Bregman ADMM (BADMM) based on the original one (Wang and Banerjee, 2014) to solve the LP. In this method, all subproblems have closed-form solutions and hence can be solved efficiently.

Promising numerical performance was also reported in (Ye et al., 2017). However, this modified Bregman ADMM does not have a convergence guarantee so far.

In this paper, we also consider the LP as a constrained convex problem with multiple blocks of variables and develop an efficient method to solve its dual LP *without* introducing the entropy regularization to modify the objective function. Here, we should mention that although introducing the entropy regularization can give a certain 'smooth' solution (that may be favorable in some learning tasks) and lead to the efficient method IBP, it also introduces some blurring in transport plans (see Figure 1(b)), which may be undesirable in many other applications. The blurred transport plan can be hard to use for other purposes, for example, the recovery of non-mass splitting transport plan (Borgwardt, 2020). In contrast, as discussed in (Borgwardt, 2020), an exact barycenter computed from the non-regularized LP can have several favorable properties. Therefore, we believe it is important to have an efficient algorithm that can faithfully solve the original LP. Moreover, the non-regularization-based method can naturally avoid the numerical issues caused by the entropy regularization and thus it is numerically more stable.

Our method is a convergent 3-block ADMM that is designed based on recent progresses in research on convergent multi-block ADMM-type methods for solving convex composite conic programming; see (Chen et al., 2017; Li et al., 2016). It is well known that the classical ADMM was originally proposed to solve a convex problem that contains 2 blocks of variables and a coupling linear equality constraint (Gabay and Mercier, 1976; Glowinski and Marroco, 1975). Specifically, consider

$$\min_{\boldsymbol{x}_1\in\mathbb{R}^{n_1},\,\boldsymbol{x}_2\in\mathbb{R}^{n_2}} \; f_1(\boldsymbol{x}_1)+f_2(\boldsymbol{x}_2) \quad \text{s.t.} \quad \mathcal{A}_1(\boldsymbol{x}_1)+\mathcal{A}_2(\boldsymbol{x}_2)=\boldsymbol{b}, \tag{1}$$

where $f_1:\mathbb{R}^{n_1}\to(-\infty,\infty]$ and $f_2:\mathbb{R}^{n_2}\to(-\infty,\infty]$ are proper closed convex functions, $\mathcal{A}_1:\mathbb{R}^{n_1}\to\mathbb{R}^m$ and $\mathcal{A}_2:\mathbb{R}^{n_2}\to\mathbb{R}^m$ are linear operators, $\boldsymbol{b}\in\mathbb{R}^m$ is a given vector. The iterative scheme of the ADMM for solving problem (1) is given as follows:

$$\begin{cases} \boldsymbol{x}_1^{k+1}\in\arg\min\limits_{\boldsymbol{x}_1\in\mathbb{R}^{n_1}}\left\{\mathcal{L}_\beta(\boldsymbol{x}_1,\,\boldsymbol{x}_2^k,\,\boldsymbol{\lambda}^k)\right\}, \\ \boldsymbol{x}_2^{k+1}\in\arg\min\limits_{\boldsymbol{x}_2\in\mathbb{R}^{n_2}}\left\{\mathcal{L}_\beta(\boldsymbol{x}_1^{k+1},\,\boldsymbol{x}_2,\,\boldsymbol{\lambda}^k)\right\}, \\ \boldsymbol{\lambda}^{k+1}=\boldsymbol{\lambda}^k+\tau\beta\big(\mathcal{A}_1(\boldsymbol{x}_1^{k+1})+\mathcal{A}_2(\boldsymbol{x}_2^{k+1})-\boldsymbol{b}\big), \end{cases}$$

where $\tau\in(0,\frac{\sqrt{5}+1}{2})$ is the dual step-size and $\mathcal{L}_\beta$ is the augmented Lagrangian function for (1) defined as

$$\mathcal{L}_\beta(\boldsymbol{x}_1,\,\boldsymbol{x}_2,\,\boldsymbol{\lambda}):=f_1(\boldsymbol{x}_1)+f_2(\boldsymbol{x}_2)+\langle\boldsymbol{\lambda},\,\mathcal{A}_1(\boldsymbol{x}_1)+\mathcal{A}_2(\boldsymbol{x}_2)-\boldsymbol{b}\rangle+\tfrac{\beta}{2}\|\mathcal{A}_1(\boldsymbol{x}_1)+\mathcal{A}_2(\boldsymbol{x}_2)-\boldsymbol{b}\|^2$$

with $\beta>0$ being the penalty parameter. Under some mild conditions, the sequence $\{(\boldsymbol{x}_1^k,\,\boldsymbol{x}_2^k)\}$ generated by the above scheme can be shown to converge to an optimal solution of problem (1). The above 2-block ADMM can be simply extended to a multi-block ADMM of the sequential Gauss-Seidel order for solving a convex problem with more than 2 blocks of variables. However, it has been shown in (Chen et al., 2016) that such a directly extended ADMM may not converge when applied to a problem with 3 *or* more blocks of variables. This has motivated many researchers to develop various convergent variants of the ADMM for convex problems with more than 2 blocks of variables; see, for example, (Chen et al., 2017, 2019; He et al., 2012; Li et al., 2015, 2016; Sun et al., 2015). Among them, the Schur complement based convergent semi-proximal ADMM (sPADMM) was proposed by Li et al. (2016) to solve a large class of linearly constrained convex problems with multiple blocks of variables, whose objective can be the sum of two proper closed convex functions and a finite number of convex quadratic or linear functions. This method modified the original ADMM by performing one more *forward Gauss-Seidel sweep* after updating the block of variables corresponding to the nonsmooth function in the objective. With this novel strategy, Li et al. (2016) showed that their method can be reformulated as a 2-block sPADMM with specially designed semi-proximal terms and its convergence is thus guaranteed from that of the 2-block sPADMM; see (Fazel et al., 2013, Appendix B). Later, this method was generalized to the inexact symmetric Gauss-Seidel based ADMM (sGS-ADMM) for more general convex problems (Chen et al., 2017; Li et al., 2018). The numerical results reported in (Chen et al., 2017; Li et al., 2016, 2018) also showed that the sGS-ADMM always performs much better than the possibly non-convergent directly extended ADMM. In addition, as the sGS-ADMM is equivalent to a 2-block sPADMM with specially designed proximal terms, the linear convergence rate of the sGS-ADMM can also be derived based on the linear convergence rate of the 2-block sPADMM under some mild conditions; more details can be found in (Han et al., 2018, Section 4.1).

Motivated by the above studies, in this paper, we adapt the sGS-ADMM to compute a Wasserstein barycenter by solving the dual problem of the original primal LP. The contributions of this paper are listed as follows:

1. We derive the dual problem of the original primal LP and characterize the properties of their optimal solutions; see Proposition 5. The resulting dual problem is our target problem, which is reformulated as a linearly constrained convex problem containing 3 blocks of variables with a carefully delineated separable structure designed for efficient computations. We should emphasize again that we do not introduce the entropic or quadratic regularization to modify the LP so as to make it computationally more tractable. This is in contrast to many existing works that primarily focus on solving an approximation of the original LP arising from optimal transport related problems; see, for example, (Benamou et al., 2015; Cuturi, 2013; Cuturi and Doucet, 2014; Dessein et al., 2018; Essid and Solomon, 2018).

2. We apply the sGS-ADMM to solve the resulting dual problem and analyze its global convergence as well as its global linear convergence rate without any condition; see Theorems 6 and 7. As a critical component of the paper, we also develop essential numerical strategies to show how all the subproblems in our method can be solved efficiently and that the subproblems at each step can be computed in parallel. This makes our sGS-ADMM highly suitable for computing Wasserstein barycenters on a large-scale data set.

3. We conduct rigorous numerical experiments on synthetic data sets and image data sets to evaluate the performance of our sGS-ADMM in comparison to existing state-of-the-art methods (IBP and BADMM) and the highly powerful commercial solver Gurobi. The computational results show that our sGS-ADMM is highly competitive

compared to IBP and BADMM, and is also able to outperform Gurobi in terms of the computational time for solving large-scale LPs arising from Wasserstein barycenter problems.

The rest of this paper is organized as follows. In Section 2, we describe the basic problem of computing Wasserstein barycenters and derive its dual problem. In Section 3, we adapt the sGS-ADMM to solve the resulting dual problem and present the efficient implementations for each step that are crucial in making our method competitive. The convergence analysis of the sGS-ADMM is presented in Section 4. Finally, numerical results are presented in Section 5, with some concluding remarks given in Section 6.

Notation and Preliminaries. In this paper, we present scalars, vectors and matrices in lower case letters, bold lower case letters and upper case letters, respectively. We use $\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^n_+$ and $\mathbb{R}^{m\times n}$ to denote the set of real numbers, n-dimensional real vectors, n-dimensional real vectors with nonnegative entries and $m\times n$ real matrices, respectively. For a vector $\boldsymbol{x}$, x_i denotes its i-th entry, $\|\boldsymbol{x}\|$ denotes its Euclidean norm, $\|\boldsymbol{x}\|_p$ denotes its ℓ_p-norm ($p\geq 1$) defined by $\|\boldsymbol{x}\|_p := (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}}$ and $\|\boldsymbol{x}\|_T := \sqrt{\langle \boldsymbol{x},\, T\boldsymbol{x}\rangle}$ denotes its weighted norm associated with the symmetric positive semidefinite matrix T. For a matrix X, x_{ij} denotes its (i,j)-th entry, $X_{i:}$ denotes its i-th row, $X_{:j}$ denotes its j-th column, $\|X\|_F$ denotes its Fröbenius norm and $\mathrm{vec}(X)$ denotes the vectorization of X. We also use $\boldsymbol{x}\geq 0$ and $X\geq 0$ to denote $x_i\geq 0$ for all i and $x_{ij}\geq 0$ for all (i,j). The identity matrix of size $n\times n$ is denoted by I_n. For any $X_1\in\mathbb{R}^{m\times n_1}$ and $X_2\in\mathbb{R}^{m\times n_2}$, $[X_1,X_2]\in\mathbb{R}^{m\times(n_1+n_2)}$ denotes the matrix obtained by horizontally concatenating X_1 and X_2. For any $Y_1\in\mathbb{R}^{m_1\times n}$ and $Y_2\in\mathbb{R}^{m_2\times n}$, $[Y_1;Y_2]\in\mathbb{R}^{(m_1+m_2)\times n}$ denotes the matrix obtained by vertically concatenating Y_1 and Y_2. For any $X\in\mathbb{R}^{m\times n}$ and $Y\in\mathbb{R}^{m'\times n'}$, the Kronecker product $X\otimes Y$ is defined as

$$X\otimes Y = \begin{bmatrix} x_{11}Y & \cdots & x_{1n}Y \\ \vdots & & \vdots \\ x_{m1}Y & \cdots & x_{mn}Y \end{bmatrix}.$$

For an extended-real-valued function $f:\mathbb{R}^n\to[-\infty,\infty]$, we say that it is *proper* if $f(\boldsymbol{x})>-\infty$ for all $\boldsymbol{x}\in\mathbb{R}^n$ and its domain $\mathrm{dom}\,f := \{\boldsymbol{x}\in\mathbb{R}^n : f(\boldsymbol{x})<\infty\}$ is nonempty. A proper function f is said to be closed if it is lower semicontinuous. Assume that $f:\mathbb{R}^n\to(-\infty,\infty]$ is a proper and closed convex function. The subdifferential of f at $\boldsymbol{x}\in\mathrm{dom}\,f$ is defined by $\partial f(\boldsymbol{x}) := \{\boldsymbol{d}\in\mathbb{R}^n : f(\boldsymbol{y})\geq f(\boldsymbol{x})+\langle \boldsymbol{d},\,\boldsymbol{y}-\boldsymbol{x}\rangle,\ \forall\,\boldsymbol{y}\in\mathbb{R}^n\}$ and its conjugate function $f^*:\mathbb{R}^n\to(-\infty,\infty]$ is defined by $f^*(\boldsymbol{y}) := \sup\{\langle \boldsymbol{y},\,\boldsymbol{x}\rangle - f(\boldsymbol{x}) : \boldsymbol{x}\in\mathbb{R}^n\}$. For any $\boldsymbol{x}$ and $\boldsymbol{y}$, it follows from (Rockafellar, 1970, Theorem 23.5) that

$$\boldsymbol{y}\in\partial f(\boldsymbol{x}) \iff \boldsymbol{x}\in\partial f^*(\boldsymbol{y}). \tag{2}$$

For any $\nu>0$, the proximal mapping of νf at $\boldsymbol{y}$ is defined by

$$\mathrm{Prox}_{\nu f}(\boldsymbol{y}) := \arg\min_{\boldsymbol{x}} \left\{ f(\boldsymbol{x}) + \frac{1}{2\nu}\|\boldsymbol{x}-\boldsymbol{y}\|^2 \right\}.$$

For a closed convex set $\mathcal{X}\subseteq\mathbb{R}^n$, its indicator function $\delta_{\mathcal{X}}$ is defined by $\delta_{\mathcal{X}}(\boldsymbol{x})=0$ if $\boldsymbol{x}\in\mathcal{X}$ and $\delta_{\mathcal{X}}(\boldsymbol{x})=+\infty$ otherwise. Moreover, we use $\mathrm{Pr}_{\mathcal{X}}(\boldsymbol{y})$ to denote the projection of $\boldsymbol{y}$ onto a closed convex set $\mathcal{X}$. It is easy to see that $\mathrm{Pr}_{\mathcal{X}}(\cdot)\equiv\mathrm{Prox}_{\delta_{\mathcal{X}}}(\cdot)$.

In the following, a discrete probability distribution $\mathcal{P}$ with finite support points is specified by $\{(a_i,\, \boldsymbol{q}_i) \in \mathbb{R}_+ \times \mathbb{R}^d : i = 1, \cdots, m\}$, where $\{\boldsymbol{q}_1, \cdots, \boldsymbol{q}_m\}$ are the support points or vectors and $\{a_1, \cdots, a_m\}$ are the associated probabilities or weights satisfying $\sum_{i=1}^m a_i = 1$ and $a_i \geq 0$, $i = 1, \cdots, m$. We also use $\Xi^p(\mathbb{R}^d)$ to denote the set of all discrete probability distributions on $\mathbb{R}^d$ with finite p-th moment.

2. Problem Statement

In this section, we briefly recall the Wasserstein distance and describe the problem of computing a Wasserstein barycenter for a set of discrete probability distributions with finite support points. We refer interested readers to (Villani, 2008, Chapter 6) for more details on the Wasserstein distance and to (Agueh and Carlier, 2011; Anderes et al., 2016) for more details on the Wasserstein barycenter.

Given two discrete distributions $\mathcal{P}^{(u)} = \{(a_i^{(u)},\, \boldsymbol{q}_i^{(u)}) : i = 1, \cdots, m_u\}$ and $\mathcal{P}^{(v)} = \{(a_i^{(v)},\, \boldsymbol{q}_i^{(v)}) : i = 1, \cdots, m_v\}$, the p-Wasserstein distance between $\mathcal{P}^{(u)}$ and $\mathcal{P}^{(v)}$ is defined by

$$\mathcal{W}_p(\mathcal{P}^{(u)},\, \mathcal{P}^{(v)}) := \sqrt[p]{\mathrm{v}^*},$$

where $p \geq 1$ (commonly chosen to be 1 *or* 2) and v^* is the optimal objective value of the following linear programming problem:

$$\mathrm{v}^* := \min_{\pi_{ij} \geq 0} \left\{ \sum_i^{m_u} \sum_j^{m_v} \pi_{ij} \|\boldsymbol{q}_i^{(u)} - \boldsymbol{q}_j^{(v)}\|_p^p \ : \ \begin{array}{ll} \sum_{i=1}^{m_u} \pi_{ij} = a_j^{(v)}, & j = 1, \cdots, m_v \\ \sum_{j=1}^{m_v} \pi_{ij} = a_i^{(u)}, & i = 1, \cdots, m_u \end{array} \right\}.$$

Then, given a set of discrete probability distributions $\{\mathcal{P}^{(t)}\}_{t=1}^N$ with $\mathcal{P}^{(t)} = \{(a_i^{(t)},\, \boldsymbol{q}_i^{(t)}) : i = 1, \cdots, m_t\}$, a p-Wasserstein barycenter $\mathcal{P} := \{(w_i,\, \boldsymbol{x}_i) : i = 1, \cdots, m\}$ with m support points is an optimal solution of the following problem

$$\min_{\mathcal{P} \in \Xi^p(\mathbb{R}^d)} \ \sum_{t=1}^N \gamma_t \big(\mathcal{W}_p(\mathcal{P},\, \mathcal{P}^{(t)})\big)^p$$

for given weights $(\gamma_1, \cdots, \gamma_N)$ satisfying $\sum_{t=1}^N \gamma_t = 1$ and $\gamma_t > 0$, $t = 1, \cdots, N$. It is worth noting that the number of support points of the true barycenter is generally unknown. In theory, for $p = 2$, there exists a sparse barycenter whose number of support points is upper bounded by $\sum_{t=1}^N m_t - N + 1$; see (Anderes et al., 2016, Theorem 2). In practice, one usually chooses m by experience and sets a value that is not less than m_t for $t = 1, \cdots, N$. Clearly, a larger m would lead to a larger problem size and hence may require more computational cost, as observed from our experiments. Since the Wasserstein distance itself is defined by a LP, the above problem is then a two-stage optimization problem. Using the definition with some simple manipulations, one can equivalently rewrite the above problem as

$$\begin{array}{lll} \displaystyle\min_{\boldsymbol{w},\, X,\, \{\Pi^{(t)}\}} & \sum_{t=1}^N \langle \gamma_t \mathcal{D}(X,\, Q^{(t)}),\, \Pi^{(t)} \rangle & \\ \text{s.t.} & \Pi^{(t)} \boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0, \quad t = 1, \cdots, N, & \quad (3) \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, & \end{array}$$

where

- $\boldsymbol{e}_{m_t}$ (resp. $\boldsymbol{e}_m$) denotes the m_t (resp. m) dimensional vector with all entries being 1;
- $\boldsymbol{w} := (w_1, \cdots, w_m)^\top \in \mathbb{R}^m_+$, $X := [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m] \in \mathbb{R}^{d\times m}$;
- $\boldsymbol{a}^{(t)} := (a^{(t)}_1, \cdots, a^{(t)}_{m_t})^\top \in \mathbb{R}^{m_t}_+$, $Q^{(t)} := \big[\boldsymbol{q}^{(t)}_1, \cdots, \boldsymbol{q}^{(t)}_{m_t}\big] \in \mathbb{R}^{d\times m_t}$ for $t = 1, \cdots, N$;
- $\Pi^{(t)} = \big[\,\pi^{(t)}_{ij}\,\big] \in \mathbb{R}^{m\times m_t}$, $\mathcal{D}(X, Q^{(t)}) := \big[\,\|\boldsymbol{x}_i - \boldsymbol{q}^{(t)}_j\|^p_p\,\big] \in \mathbb{R}^{m\times m_t}$ for $t = 1, \cdots, N$.

Note that problem (3) is a nonconvex problem, where one needs to find the optimal support 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 $\{\mathcal{P}^{(t)}\}^N_{t=1}$. Thus, one only needs to find the weight vector $\boldsymbol{w}$ of a barycenter. In view of this, from now on, we assume that the support X is given. Consequently, problem (3) reduces to the following problem:

$$\begin{array}{cl} \min\limits_{\boldsymbol{w},\,\{\Pi^{(t)}\}} & \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0, \quad t = 1, \cdots, N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{4}$$

where $D^{(t)}$ denotes $\gamma_t \mathcal{D}(X, Q^{(t)})$ for simplicity[1]. This is also the main problem studied in (Benamou et al., 2015; Borgwardt, 2020; Carlier et al., 2015; Claici et al., 2018; Cuturi and Doucet, 2014; Cuturi and Peyré, 2016; Oberman and Ruan, 2015; Schmitzer, 2019; Uribe et al., 2018; Ye and Li, 2014; Ye et al., 2017) for the computation of Wasserstein barycenters. Moreover, one can easily see that problem (4) is indeed a large-scale LP containing $(m + m\sum_{t=1}^N m_t)$ nonnegative variables and $(Nm + \sum_{t=1}^N m_t + 1)$ equality constraints. For $N = 100$, $m = 1000$ and $m_t = 1000$ for all $t = 1, \cdots, N$, such LP has about 10^8 nonnegative variables and 2×10^5 equality constraints.

Remark 1 (Practical computational consideration when $\boldsymbol{a}^{(t)}$ is sparse) *Note that any feasible point $(\boldsymbol{w}, \{\Pi^{(t)}\})$ of problem (4) must satisfy $(\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)}$ and $\Pi^{(t)} \geq 0$ for any $t = 1, \cdots, N$. This implies that if $a^{(t)}_j = 0$ for some $1 \leq j \leq m_t$ and $1 \leq t \leq N$, then $\pi^{(t)}_{ij} = 0$ for all $1 \leq i \leq m$, i.e., all entries in the j-th column of $\Pi^{(t)}$ are zeros. Based on this fact, one can verify the following statements.*

- *For any optimal solution $(\boldsymbol{w}^*, \{\Pi^{(t),*}\})$ of problem (4), the point $(\boldsymbol{w}^*, \{\Pi^{(t),*}_{\mathcal{J}_t}\})$ is also an optimal solution of the following problem*

$$\begin{array}{cl} \min\limits_{\boldsymbol{w},\,\{\widehat{\Pi}^{(t)}\}} & \sum_{t=1}^N \langle D^{(t)},\, \widehat{\Pi}^{(t)}\rangle \\ \text{s.t.} & \widehat{\Pi}^{(t)}\boldsymbol{e}_{m'_t} = \boldsymbol{w},\ (\widehat{\Pi}^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)}_{\mathcal{J}_t},\ \widehat{\Pi}^{(t)} \geq 0, \quad t = 1, \cdots, N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{5}$$

where $\mathcal{J}_t$ denotes the support set of $\boldsymbol{a}^{(t)}$, i.e., $\mathcal{J}_t := \{\, j : a^{(t)}_j \neq 0\,\}$, m'_t denotes the cardinality of $\mathcal{J}_t$, $\boldsymbol{a}^{(t)}_{\mathcal{J}_t} \in \mathbb{R}^{m'_t}$ denotes the subvector of $\boldsymbol{a}^{(t)}$ obtained by selecting the

1. Our method presented later actually can solve problem (4) for any given matrices $D^{(1)}, \cdots, D^{(N)}$.

entries indexed by $\mathcal{J}_t$ *and* $\Pi^{(t),*}_{\mathcal{J}_t} \in \mathbb{R}^{m\times m'_t}$ *denotes the submatrix of* $\Pi^{(t),*}$ *obtained by selecting the columns indexed by* $\mathcal{J}_t$*.*

- *For any optimal solution* $(\boldsymbol{w}^*, \{\widehat{\Pi}^{(t),*}\})$ *of problem* (5)*, the point* $(\boldsymbol{w}^*, \{\Pi^{(t),*}\})$ *obtained by setting* $\Pi^{(t),*}_{\mathcal{J}_t} = \widehat{\Pi}^{(t),*}$ *and* $\Pi^{(t),*}_{\mathcal{J}_t^c} = 0$ *is also an optimal solution of problem* (4)*, where* $\mathcal{J}_t^c := \{\, j : a_j^{(t)} = 0 \,\}$*.*

Therefore, one can obtain an optimal solution of problem (4) *by computing an optimal solution of problem* (5)*. Note that the size of problem* (5) *can be much smaller than that of problem* (4) *when each* $\boldsymbol{a}^{(t)}$ *is sparse, i.e.,* $m'_t \ll m_t$*. Thus, solving problem* (5) *can reduce the computational cost and save memory in practice. Since problem* (5) *takes the same form as problem* (4)*, we only consider problem* (4) *in the following.*

For notational simplicity, let $\Delta_m := \{\boldsymbol{w} \in \mathbb{R}^m : \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0\}$ and δ_+^t be the indicator function over $\{\Pi^{(t)} \in \mathbb{R}^{m\times m_t} : \Pi^{(t)} \geq 0\}$ for each $t = 1, \cdots, N$. By enforcing the constraints $\boldsymbol{w} \in \Delta_m$ and $\Pi^{(t)} \geq 0$, $t = 1, \cdots, N$ via adding the corresponding indicator functions in the objective, problem (4) can be equivalently written as

$$\begin{array}{cl} \min\limits_{\boldsymbol{w},\,\{\Pi^{(t)}\}} & \delta_{\Delta_m}(\boldsymbol{w}) + \sum_{t=1}^N \delta_+^t(\Pi^{(t)}) + \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \ t = 1, \cdots, N. \end{array} \tag{6}$$

We next derive the dual problem of (6) (hence (4)). To this end, we write down the Lagrangian function associated with (6) as follows:

$$\begin{aligned} \Upsilon\big(\boldsymbol{w}, \{\Pi^{(t)}\}; \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big) &:= \delta_{\Delta_m}(\boldsymbol{w}) + \sum_{t=1}^N \delta_+^t(\Pi^{(t)}) + \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ &+ \sum_{t=1}^N \langle \boldsymbol{y}^{(t)},\, \Pi^{(t)}\boldsymbol{e}_{m_t} - \boldsymbol{w}\rangle + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, (\Pi^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)}\rangle, \end{aligned} \tag{7}$$

where $\boldsymbol{y}^{(t)} \in \mathbb{R}^m$, $\boldsymbol{z}^{(t)} \in \mathbb{R}^{m_t}$, $t = 1, \cdots, N$ are multipliers. Then, the dual problem of (6) is given by

$$\max_{\{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}}\ \min_{\boldsymbol{w}, \{\Pi^{(t)}\}}\ \Upsilon\big(\boldsymbol{w}, \{\Pi^{(t)}\}; \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big). \tag{8}$$

Observe that

$$\begin{aligned} &\min_{\boldsymbol{w},\,\{\Pi^{(t)}\}}\ \Upsilon\big(\boldsymbol{w}, \{\Pi^{(t)}\}; \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big) \\ &= \min_{\boldsymbol{w},\,\{\Pi^{(t)}\}} \left\{ \begin{array}{l} \delta_{\Delta_m}(\boldsymbol{w}) - \langle \sum_{t=1}^N \boldsymbol{y}^{(t)},\, \boldsymbol{w}\rangle + \sum_{t=1}^N \big(\delta_+^t(\Pi^{(t)}) + \langle D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top,\, \Pi^{(t)}\rangle\big) \\ \quad - \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle \end{array} \right\} \\ &= \begin{cases} -\,\delta^*_{\Delta_m}\big(\sum_{t=1}^N \boldsymbol{y}^{(t)}\big) - \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle, & \text{if } D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top \geq 0,\ t = 1, \cdots, N, \\ -\,\infty, & \text{otherwise}, \end{cases} \end{aligned}$$

where $\delta^*_{\Delta_m}$ is the conjugate of δ_{Δ_m}. Thus, (8) is equivalent to

$$\begin{array}{cl}\min\limits_{\{\boldsymbol{y}^{(t)}\},\,\{\boldsymbol{z}^{(t)}\}} & \delta^*_{\Delta_m}\big(\sum_{t=1}^N \boldsymbol{y}^{(t)}\big) + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle \\ \text{s.t.} & D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top \geq 0, \quad t = 1, \cdots, N.\end{array}$$

By introducing auxiliary variables $\boldsymbol{u}, V^{(1)}, \cdots, V^{(N)}$, we can further reformulate the above problem as

$$\begin{array}{cl}\min\limits_{\boldsymbol{u},\,\{V^{(t)}\},\,\{\boldsymbol{y}^{(t)}\},\,\{\boldsymbol{z}^{(t)}\}} & \delta^*_{\Delta_m}(\boldsymbol{u}) + \sum_{t=1}^N \delta^t_+(V^{(t)}) + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle \\ \text{s.t.} & \sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u} = 0, \\ & V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top = 0, \quad t = 1, \cdots, N.\end{array} \tag{9}$$

Note that problem (9) can be viewed as a linearly constrained convex problem with 3 blocks of variables grouped as $\big(\boldsymbol{u},\, \{V^{(t)}\}\big)$, $\{\boldsymbol{y}^{(t)}\}$ and $\{\boldsymbol{z}^{(t)}\}$, whose objective is nonsmooth only with respect to $\big(\boldsymbol{u},\, \{V^{(t)}\}\big)$ and linear with respect to the other two. Thus, this problem exactly falls into the class of convex problems for which the sGS-ADMM is applicable; see (Chen et al., 2017; Li et al., 2016). Then, it is natural to adapt the sGS-ADMM for solving problem (9), which is presented in the next section.

Remark 2 (2-block ADMM for solving (4)**)** *It is worth noting that one can also apply the 2-block ADMM to solve the primal problem* (4) *by introducing some proper auxiliary variables. For example, one can consider the following equivalent reformulation of* (4)*:*

$$\begin{array}{cl}\min\limits_{\boldsymbol{w},\{\Pi^{(t)}\},\{\Gamma^{(t)}\}} & \delta_{\Delta_m}(\boldsymbol{w}) + \sum_{t=1}^N \delta_{\Delta_{\Pi^{(t)}}}(\Pi^{(t)}) + \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)} = \Gamma^{(t)}, \quad \Gamma^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w}, \quad t = 1, \cdots, N,\end{array}$$

where $\Delta_{\Pi^{(t)}} := \{\Pi^{(t)} \in \mathbb{R}^{m\times m_t} : (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0\}$*. Then, the 2-block ADMM can be readily applied with* $(\boldsymbol{w}, \Pi^{(1)}, \cdots, \Pi^{(N)})$ *being one block and* $(\Gamma^{(1)}, \cdots, \Gamma^{(N)})$ *as the other. This 2-block ADMM avoids the need to solve quadratic programming subproblems and hence is more efficient than the one used in (Ye and Li, 2014). However, it needs to compute the projection onto the* m*-dimensional simplex* $(1 + \sum_{t=1}^N m_t)$ *times when solving the* $(\boldsymbol{w}, \Pi^{(1)}, \cdots, \Pi^{(N)})$*-subproblem in each iteration. This is still time-consuming when* N *or* m_t *is large. Thus, this 2-block ADMM is also not efficient enough for solving large-scale problems. In addition, we have adapted the 2-block ADMM for solving other reformulations of* (4)*, but they all perform worse than our sGS-ADMM to be presented later. Hence, we will no longer consider ADMM-type methods for solving the primal problem* (4) *or its equivalent variants in this paper.*

3. sGS-ADMM for Computing Wasserstein Barycenters

In this section, we present the sGS-ADMM for solving problem (9). First, we write down the Lagrangian function associated with (9) as follows:

$$\begin{array}{l}\widetilde{\Upsilon}\big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}, \{\Lambda^{(t)}\}\big) \\ = \delta^*_{\Delta_m}(\boldsymbol{u}) + \sum_{t=1}^N \delta^t_+(V^{(t)}) + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle + \langle \boldsymbol{\lambda},\, \sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}\rangle \\ \quad + \sum_{t=1}^N \langle \Lambda^{(t)},\, V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\rangle,\end{array} \tag{10}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$, $\Lambda^{(t)} \in \mathbb{R}^{m\times m_t}$, $t = 1, \cdots, N$ are multipliers. Then, the augmented Lagrangian function associated with (9) is

$$\begin{aligned}
&\mathcal{L}_\beta\big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}, \{\Lambda^{(t)}\}\big) \\
&= \widetilde{\Upsilon}\big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}, \{\Lambda^{(t)}\}\big) + \tfrac{\beta}{2}\|\textstyle\sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}\|^2 \\
&\quad + \tfrac{\beta}{2}\textstyle\sum_{t=1}^N \|V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\|_F^2,
\end{aligned}$$

where $\beta > 0$ is the penalty parameter. The sGS-ADMM for solving (9) is now readily presented in Algorithm 1.

```
Algorithm 1 sGS-ADMM for solving (9)
Input: the penalty parameter $\beta > 0$, and the initialization $\boldsymbol{u}^0 \in \mathbb{R}^m$, $\boldsymbol{\lambda}^0 \in \mathbb{R}^m$, $\boldsymbol{y}^{(t),0} \in \mathbb{R}^m$, $\boldsymbol{z}^{(t),0} \in \mathbb{R}^{m_t}$, $V^{(t),0} \in \mathbb{R}_+^{m\times m_t}$, $\Lambda^{(t),0} \in \mathbb{R}^{m\times m_t}$, $t = 1, \cdots, N$. Set $k = 0$.
while a termination criterion is not met, do
    Step 1. Compute
        $\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}\big) = \arg\min_{\boldsymbol{u}, \{V^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.
    Step 2a. Compute
        $\{\tilde{\boldsymbol{z}}^{(t),k+1}\} = \arg\min_{\{\boldsymbol{z}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.
    Step 2b. Compute
        $\{\boldsymbol{y}^{(t),k+1}\} = \arg\min_{\{\boldsymbol{y}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t)}\}, \{\tilde{\boldsymbol{z}}^{(t),k+1}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.
    Step 2c. Compute
        $\{\boldsymbol{z}^{(t),k+1}\} = \arg\min_{\{\boldsymbol{z}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t),k+1}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.
    Step 3. Compute
        $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \tau\beta\big(\sum_{t=1}^N \boldsymbol{y}^{(t),k+1} - \boldsymbol{u}^{k+1}\big)$,
        $\Lambda^{(t),k+1} = \Lambda^{(t),k} + \tau\beta\big(V^{(t),k+1} - D^{(t)} - \boldsymbol{y}^{(t),k+1}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t),k+1})^\top\big)$,  $t = 1, \cdots, N$,
        where $\tau \in (0, \frac{1+\sqrt{5}}{2})$ is the dual step-size that is typically set to 1.618.
end while
Output: $\boldsymbol{u}^{k+1}$, $\{V^{(t),k+1}\}$, $\{\boldsymbol{y}^{(t),k+1}\}$, $\{\boldsymbol{z}^{(t),k+1}\}$, $\boldsymbol{\lambda}^{k+1}$, $\{\Lambda^{(t),k+1}\}$.
```

We next show that all subproblems in Algorithm 1 can be solved efficiently (in fact analytically) and the subproblems in each step can also be computed in parallel. This makes our method highly suitable for solving large-scale problems. The computational details and the efficient implementations in each step are presented as follows.

Step 1. Note that $\mathcal{L}_\beta$ is actually separable with respect to $\boldsymbol{u}, V^{(1)}, \cdots, V^{(N)}$ and hence one can compute $\boldsymbol{u}^{k+1}, V^{(1),k+1}, \cdots, V^{(N),k+1}$ independently. Specifically, $\boldsymbol{u}^{k+1}$ is obtained by solving

$$\min_{\boldsymbol{u}\in\mathbb{R}^m} \Big\{ \delta^*_{\Delta_m}(\boldsymbol{u}) - \langle \boldsymbol{\lambda}^k,\, \boldsymbol{u}\rangle + \tfrac{\beta}{2}\| \textstyle\sum_{t=1}^N \boldsymbol{y}^{(t),k} - \boldsymbol{u}\|^2 \Big\}.$$

Thus, we have

$$\begin{aligned}\boldsymbol{u}^{k+1} &= \mathrm{Prox}_{\beta^{-1}\delta^*_{\Delta_m}}\big(\beta^{-1}\boldsymbol{\lambda}^k + \textstyle\sum_{t=1}^N \boldsymbol{y}^{(t),k}\big) \\ &= \big(\beta^{-1}\boldsymbol{\lambda}^k + \textstyle\sum_{t=1}^N \boldsymbol{y}^{(t),k}\big) - \beta^{-1}\,\mathrm{Prox}_{\beta\delta_{\Delta_m}}\big(\boldsymbol{\lambda}^k + \beta \textstyle\sum_{t=1}^N \boldsymbol{y}^{(t),k}\big),\end{aligned}$$

where the last equality follows from the Moreau decomposition (Bauschke and Combettes, 2011, Theorem 14.3(ii)), i.e., $\boldsymbol{x} = \mathrm{Prox}_{\nu f^*}(\boldsymbol{x}) + \nu \mathrm{Prox}_{f/\nu}(\boldsymbol{x}/\nu)$ for any $\nu > 0$ and the proximal mapping of $\beta\delta_{\Delta_m}$ can be computed efficiently by the algorithm proposed in (Condat, 2016) with the complexity of $\mathcal{O}(m)$ that is typically observed in practice. Moreover, for each $t = 1, \cdots, N$, $V^{(t),k+1}$ can be computed in parallel by solving

$$\min_{V^{(t)}} \Big\{ \delta^t_+(V^{(t)}) + \langle \Lambda^{(t),k},\, V^{(t)}\rangle + \tfrac{\beta}{2}\|V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t),k}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t),k})^\top\|_F^2 \Big\}.$$

Then, it is easy to see that

$$V^{(t),k+1} = \max\big\{\widetilde{D}^{(t),k} - \beta^{-1}\Lambda^{(t),k},\, 0\big\},$$

where $\widetilde{D}^{(t),k} := D^{(t)} + \boldsymbol{y}^{(t),k}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t),k})^\top$. Note that $\widetilde{D}^{(t),k}$ is already computed for updating $\Lambda^{(t),k}$ in the previous iteration and thus it can be reused in the current iteration. The computational complexity in this step is $\mathcal{O}(Nm + m\sum_{t=1}^N m_t)$. We should emphasize that because the matrices such as $\{\widetilde{D}^{(t),k}\}$, $\{\Lambda^{(t),k}\}$ are large and numerous, even performing simple operations such as adding two such matrices can be time-consuming. Thus, we have paid special attention to arrange the computations in each step of the sGS-ADMM so that matrices computed in one step can be reused for the next step.

Step 2a. Similarly, $\mathcal{L}_\beta$ is separable with respect to $\boldsymbol{z}^{(1)}, \cdots, \boldsymbol{z}^{(N)}$ and then one can also compute $\tilde{\boldsymbol{z}}^{(1),k+1}, \ldots, \tilde{\boldsymbol{z}}^{(N),k+1}$ in parallel. For each $t = 1, \cdots, N$, $\tilde{\boldsymbol{z}}^{(t),k+1}$ is obtained by solving

$$\min_{\boldsymbol{z}^{(t)}} \Big\{ \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle - \langle \Lambda^{(t),k},\, \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\rangle + \tfrac{\beta}{2}\|V^{(t),k+1} - D^{(t)} - \boldsymbol{y}^{(t),k}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\|_F^2 \Big\}.$$

It is easy to prove that

$$\begin{aligned}\tilde{\boldsymbol{z}}^{(t),k+1} &= \tfrac{1}{m}\big((V^{(t),k+1})^\top\boldsymbol{e}_m - (D^{(t)})^\top\boldsymbol{e}_m - \big(\boldsymbol{e}_m^\top\boldsymbol{y}^{(t),k}\big)\boldsymbol{e}_{m_t} + \beta^{-1}(\Lambda^{(t),k})^\top\boldsymbol{e}_m - \beta^{-1}\boldsymbol{a}^{(t)}\big) \\ &= \boldsymbol{z}^{(t),k} - \tfrac{1}{m}\big(\beta^{-1}\boldsymbol{a}^{(t)} + (B^{(t),k})^\top\boldsymbol{e}_m\big),\end{aligned}$$

where $B^{(t),k} := \widetilde{D}^{(t),k} - \beta^{-1}\Lambda^{(t),k} - V^{(t),k+1} = \min\{\widetilde{D}^{(t),k} - \beta^{-1}\Lambda^{(t),k},\, 0\}$. Note that $\widetilde{D}^{(t),k} - \beta^{-1}\Lambda^{(t),k}$ has already been computed in **Step 1** and hence $B^{(t),k}$ can be computed by just a simple $\min(\cdot)$ operation. We note that $\tilde{\boldsymbol{z}}^{(t),k+1}$ is computed analytically for all $t = 1, \ldots, N$, and the computational complexity in this step is $\mathcal{O}((m+1)\sum_{t=1}^{N} m_t)$.

Step 2b. In this step, one can see that $\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(N)}$ are coupled in $\mathcal{L}_\beta$ (due to the quadratic term $\frac{\beta}{2}\|\sum_{t=1}^{N}\boldsymbol{y}^{(t)} - \boldsymbol{u}^{k+1}\|^2$) and hence the problem of minimizing $\mathcal{L}_\beta$ with respect to $\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(N)}$ cannot be reduced to N separable subproblems. However, one can still compute them efficiently based on the following observation. Note that $(\boldsymbol{y}^{(1),k+1}, \cdots, \boldsymbol{y}^{(N),k+1})$ is obtained by solving

$$\min_{\boldsymbol{y}^{(1)},\cdots,\boldsymbol{y}^{(N)}} \left\{ \begin{array}{l} \Phi^k(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(N)}) := \langle \boldsymbol{\lambda}^k, \sum_{t=1}^{N}\boldsymbol{y}^{(t)}\rangle + \frac{\beta}{2}\|\sum_{t=1}^{N}\boldsymbol{y}^{(t)} - \boldsymbol{u}^{k+1}\|^2 \\ \quad + \sum_{t=1}^{N}\big(\langle -\Lambda^{(t),k}\boldsymbol{e}_{m_t},\, \boldsymbol{y}^{(t)}\rangle + \frac{\beta}{2}\|V^{(t),k+1} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\tilde{\boldsymbol{z}}^{(t),k+1})^\top\|_F^2\big) \end{array} \right\}.$$

The gradient of Φ^k with respect to $\boldsymbol{y}^{(t)}$ is

$$\begin{aligned} &\nabla_{\boldsymbol{y}^{(t)}}\Phi^k(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(N)}) \\ &= \boldsymbol{\lambda}^k + \beta\big(\textstyle\sum_{\ell=1}^{N}\boldsymbol{y}^{(\ell)} - \boldsymbol{u}^{k+1}\big) + \beta\big(-\beta^{-1}\Lambda^{(t),k} + D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\tilde{\boldsymbol{z}}^{(t),k+1})^\top - V^{(t),k+1}\big)\boldsymbol{e}_{m_t} \\ &= \beta\textstyle\sum_{\ell=1}^{N}\boldsymbol{y}^{(\ell)} + \beta m_t(\boldsymbol{y}^{(t)} - \boldsymbol{y}^{(t),k}) + \boldsymbol{\lambda}^k - \beta\boldsymbol{u}^{k+1} + \beta\big(B^{(t),k} + \boldsymbol{e}_m(\tilde{\boldsymbol{z}}^{(t),k+1} - \boldsymbol{z}^{(t),k})^\top\big)\boldsymbol{e}_{m_t} \\ &= \beta\textstyle\sum_{\ell=1}^{N}(\boldsymbol{y}^{(\ell)} - \boldsymbol{y}^{(\ell),k}) + \beta m_t(\boldsymbol{y}^{(t)} - \boldsymbol{y}^{(t),k}) + \beta\boldsymbol{h}^k + \beta\widetilde{B}^{(t),k}\boldsymbol{e}_{m_t}, \end{aligned}$$

where $\widetilde{B}^{(t),k} := B^{(t),k} + \boldsymbol{e}_m(\tilde{\boldsymbol{z}}^{(t),k+1} - \boldsymbol{z}^{(t),k})^\top$ and $\boldsymbol{h}^k := \beta^{-1}\boldsymbol{\lambda}^k - \boldsymbol{u}^{k+1} + \sum_{\ell=1}^{N}\boldsymbol{y}^{(\ell),k}$. It follows from the optimality condition, namely, $\nabla\Phi^k(\boldsymbol{y}^{(1),k+1}, \cdots, \boldsymbol{y}^{(N),k+1}) = 0$ that, for any $t = 1, \cdots, N$,

$$\textstyle\sum_{\ell=1}^{N}(\boldsymbol{y}^{(\ell),k+1} - \boldsymbol{y}^{(\ell),k}) + m_t(\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k}) + \boldsymbol{h}^k + \widetilde{B}^{(t),k}\boldsymbol{e}_{m_t} = 0. \qquad (11)$$

By dividing m_t in (11) for $t = 1, \cdots, N$, adding all resulting equations and doing some simple algebraic manipulations, one can obtain that

$$\tilde{\boldsymbol{b}}^k := \textstyle\sum_{\ell=1}^{N}(\boldsymbol{y}^{(\ell),k+1} - \boldsymbol{y}^{(\ell),k}) = -\dfrac{(\sum_{\ell=1}^{N} m_\ell^{-1})\boldsymbol{h}^k + \sum_{\ell=1}^{N} m_\ell^{-1}\widetilde{B}^{(\ell),k}\boldsymbol{e}_{m_\ell}}{1 + \sum_{\ell=1}^{N} m_\ell^{-1}}.$$

Then, using this equality and (11), we have

$$\boldsymbol{y}^{(t),k+1} = \boldsymbol{y}^{(t),k} - \tfrac{1}{m_t}\big(\tilde{\boldsymbol{b}}^k + \boldsymbol{h}^k + \widetilde{B}^{(t),k}\boldsymbol{e}_{m_t}\big), \quad t = 1, \cdots, N.$$

Observe that we can compute $\boldsymbol{y}^{(t),k+1}$ analytically for $t = 1, \cdots, N$. In the above computations, one can first compute $\widetilde{B}^{(t),k}\boldsymbol{e}_{m_t}$ in parallel for $t = 1, \cdots, N$ to obtain $\tilde{\boldsymbol{b}}^k$. Then, $\boldsymbol{y}^{(t),k+1}$ can be computed in parallel for $t = 1, \cdots, N$. By using the updating formula for $\tilde{\boldsymbol{z}}^{(t),k+1}$ in **Step 2a**, we have that $\widetilde{B}^{(t),k}\boldsymbol{e}_{m_t} = B^{(t),k}\boldsymbol{e}_{m_t} - \frac{1}{m}\boldsymbol{e}_m\big(\boldsymbol{e}_m^T B^{(t),k}\boldsymbol{e}_{m_t} + \beta^{-1}\langle \boldsymbol{e}_{m_t},\, \boldsymbol{a}^{(t)}\rangle\big)$. Thus, there is no need to form $\widetilde{B}^{(t),k}$ explicitly. The computational complexity in this step is $\mathcal{O}(Nm + m\sum_{t=1}^{N} m_t)$.

Step 2c. Similar to **Step 2a**, for each $t = 1, \cdots, N$, $\boldsymbol{z}^{(t),k+1}$ can be obtained independently by solving

$$\min_{\boldsymbol{z}^{(t)}} \left\{ \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)} \rangle - \langle \Lambda^{(t),k},\, \boldsymbol{e}_m (\boldsymbol{z}^{(t)})^\top \rangle + \tfrac{\beta}{2} \| V^{(t),k+1} - D^{(t)} - \boldsymbol{y}^{(t),k+1} \boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m (\boldsymbol{z}^{(t)})^\top \|_F^2 \right\}$$

and it is easy to show that

$$\begin{aligned} \boldsymbol{z}^{(t),k+1} &= \boldsymbol{z}^{(t),k} - \tfrac{1}{m} (\beta^{-1} \boldsymbol{a}^{(t)} + (C^{(t),k})^\top \boldsymbol{e}_m) \\ &= \boldsymbol{z}^{(t),k} - \tfrac{1}{m} (\beta^{-1} \boldsymbol{a}^{(t)} + (B^{(t),k} + (\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k}) \boldsymbol{e}_{m_t}^\top)^\top \boldsymbol{e}_m) \\ &= \tilde{\boldsymbol{z}}^{(t),k+1} - \tfrac{1}{m} \big((\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k})^\top \boldsymbol{e}_m \big) \boldsymbol{e}_{m_t}, \end{aligned}$$

where $C^{(t),k} := D^{(t)} + \boldsymbol{y}^{(t),k+1} \boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m (\boldsymbol{z}^{(t),k})^\top - \beta^{-1} \Lambda^{(t),k} - V^{(t),k+1} = B^{(t),k} + (\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k}) \boldsymbol{e}_{m_t}^\top$. In light of the above, one can also compute $\boldsymbol{z}^{(t),k+1}$ efficiently. The computational complexity in this step is $\mathcal{O}(Nm + \sum_{t=1}^N m_t)$, which is much smaller than the costs in **Step 2a** and **Step 2b**.

From the above, together with the update of multipliers in **Step 3**, one can see that the main computational complexity of our sGS-ADMM at each iteration is $\mathcal{O}(m \sum_{t=1}^N m_t)$.

Remark 3 (Comments on Step 2a–2c in Algorithm 1) *Comparing with the directly extended ADMM, our sGS-ADMM in Algorithm 1 just has one more update of* $\{\tilde{\boldsymbol{z}}^{(t),k+1}\}$ *in* **Step 2a**. *This step is actually the key to guarantee the convergence of the algorithm. We shall see in the next section that computing* $\big(\{\boldsymbol{y}^{(t),k+1}\},\, \{\boldsymbol{z}^{(t),k+1}\}\big)$ *from* **Step 2a–2c** *is indeed equivalent to minimizing* $\mathcal{L}_\beta$ *plus a special proximal term simultaneously with respect to* $\big(\{\boldsymbol{y}^{(t)}\},\, \{\boldsymbol{z}^{(t)}\}\big)$. *Moreover, the reader may have observed that instead of computing* $\{\boldsymbol{y}^{(t),k+1}\}$ *and* $\{\boldsymbol{z}^{(t),k+1}\}$ *sequentially as in* **Step 2a–2c**, *one can also compute* $(\{\boldsymbol{y}^{(t),k+1}\}, \{\boldsymbol{z}^{(t),k+1}\})$ *simultaneously in one step by solving a huge linear system of equations of dimension* $mN + \sum_{t=1}^N m_t$. *Unfortunately, for the latter approach, the computation of the solution would require the Cholesky factorization of a huge coefficient matrix, and this approach is not practically viable. In contrast, for our approach in* **Step 2a-2c**, *we have seen that the solutions can be computed analytically without the need to perform Cholesky factorizations of large coefficient matrices. This also explains why we have designed the computations as in* **Step 2a-2c**.

Remark 4 (Extension to the free support case) *We briefly discuss the case when the support points of a barycenter are not pre-specified and hence one needs to solve problem* (3) *to find a barycenter. Note that problem* (3) *can be considered as a problem with* X *being one variable block and* $(\boldsymbol{w},\, \{\Pi^{(t)}\})$ *being the other block. Then, it is natural to apply an alternating minimization method to solve* (3). *Specifically, with* X *fixed, problem* (3) *indeed reduces to problem* (4) *(hence* (6)*), and one can call our sGS-ADMM in Algorithm 1 as a subroutine to solve it efficiently. On the other hand, with* $(\boldsymbol{w},\, \{\Pi^{(t)}\})$ *fixed, problem* (3) *reduces to a simple quadratic optimization problem with respect to* X *and one can easily obtain the optimal* X^* *columnwise by computing*

$$\boldsymbol{x}_i^* = \Big(\textstyle\sum_{t=1}^N \sum_{j=1}^{m_t} \pi_{ij}^{(t)} \Big)^{-1} \textstyle\sum_{t=1}^N \sum_{j}^{m_t} \pi_{ij}^{(t)} \boldsymbol{q}_j^{(t)}, \qquad i = 1, \cdots, m.$$

In fact, this alternating minimization strategy has also been used in (Cuturi and Doucet, 2014; Ye and Li, 2014; Ye et al., 2017) to handle the free support case by using their proposed methods as subroutines.

4. Convergence Analysis

In this section, we shall establish the global linear convergence of Algorithm 1 based on the convergence results developed in (Fazel et al., 2013; Han et al., 2018; Li et al., 2016). To this end, we first write down the KKT system associated with (7) as follows:

$$\begin{aligned} 0 &\in \partial\delta_{\Delta_m}(\boldsymbol{w}) - \big(\textstyle\sum_{t=1}^{N}\boldsymbol{y}^{(t)}\big), \\ 0 &\in \partial\delta_{+}^{t}(\Pi^{(t)}) + D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^{\top} + \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^{\top}, \quad \forall\, t = 1,\cdots,N, \\ 0 &= \Pi^{(t)}\boldsymbol{e}_{m_t} - \boldsymbol{w}, \quad \forall\, t = 1,\cdots,N, \\ 0 &= (\Pi^{(t)})^{\top}\boldsymbol{e}_m - \boldsymbol{a}^{(t)}, \quad \forall\, t = 1,\cdots,N. \end{aligned} \tag{12}$$

We also write down the KKT system associated with (10) as follows:

$$\begin{aligned} 0 &\in \partial\delta_{\Delta_m}^{*}(\boldsymbol{u}) - \boldsymbol{\lambda}, \\ 0 &\in \partial\delta_{+}^{t}(V^{(t)}) + \Lambda^{(t)}, \quad \forall\, t = 1,\cdots,N, \\ 0 &= \Lambda^{(t)}\boldsymbol{e}_{m_t} - \boldsymbol{\lambda}, \quad \forall\, t = 1,\cdots,N, \\ 0 &= (\Lambda^{(t)})^{\top}\boldsymbol{e}_m - \boldsymbol{a}^{(t)}, \quad \forall\, t = 1,\cdots,N, \\ 0 &= \textstyle\sum_{t=1}^{N}\boldsymbol{y}^{(t)} - \boldsymbol{u}, \\ 0 &= V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^{\top} - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^{\top}, \quad \forall\, t = 1,\cdots,N. \end{aligned} \tag{13}$$

Then, we show the existence of optimal solutions of problems (6) and (9), and their relations in the following proposition.

Proposition 5 *The following statements hold.*

(i) *The optimal solution of problem* (6) *exists and the solution set of the KKT system* (12) *is nonempty;*

(ii) *The optimal solution of problem* (9) *exists and the solution set of the KKT system* (13) *is nonempty;*

(iii) *If* $\big(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}, \boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ *is a solution of the KKT system* (13), *then* $(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ *solves* (9) *and* $\big(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ *solves* (6).

Proof *Statement* (i). Note that (6) is equivalent to (4). Thus, we only need to show that the optimal solution of (4) exists. To this end, we first claim that the feasible set of (4) is nonempty. For simplicity, let

$$\begin{aligned} \mathcal{C}_{\rm feas} &:= \big\{(\boldsymbol{w},\, \{\Pi^{(t)}\}) : \boldsymbol{w} \in \Delta_m,\, \Pi^{(t)} \in \Omega^t(\boldsymbol{w}),\, t = 1,\cdots,N\big\}, \\ \Omega^t(\boldsymbol{w}) &:= \big\{\Pi^{(t)} \in \mathbb{R}^{m\times m_t} : \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^{\top}\boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0\big\}, \quad t = 1,\cdots,N. \end{aligned}$$

Recall that the simplex Δ_m is nonempty. Then, for any fixed $\bar{\boldsymbol{w}} \in \Delta_m$, consider the sets $\Omega^1(\bar{\boldsymbol{w}}), \cdots, \Omega^N(\bar{\boldsymbol{w}})$. For any $t = 1, \cdots, N$, since $\boldsymbol{a}^{(t)}$ is the weight vector of the discrete probability distribution $\mathcal{P}^{(t)}$, we have that $\boldsymbol{e}_{m_t}^\top \boldsymbol{a}^{(t)} = 1$. Using this fact and $\boldsymbol{e}_m^\top \bar{\boldsymbol{w}} = 1$, we have from (De Loera and Kim, 2014, Lemma 2.2) that each $\Omega^t(\bar{\boldsymbol{w}})$ is nonempty. Hence, $\mathcal{C}_{\text{feas}}$ is nonempty. Moreover, it is not hard to see that $\mathcal{C}_{\text{feas}}$ is closed and bounded. This together with the continuity of the objective function in (4) implies that the optimal solution of (4) exists. Hence, the optimal solution of (6) exists. Now, let $(\boldsymbol{w}^*, \{\Pi^{(t),*}\})$ be an optimal solution of (6). Since the set $\{(\boldsymbol{w}, \{\Pi^{(t)}\}) : \boldsymbol{w} \in \Delta_m, \Pi^{(t)} \geq 0, t = 1, \cdots, N\}$ is a convex polyhedron and all constraint functions in (6) are affine, then it follows from (Ruszczyński, 2006, Theorem 3.25) that there exist multipliers $\boldsymbol{y}^{(t),*} \in \mathbb{R}^m$, $\boldsymbol{z}^{(t),*} \in \mathbb{R}^{m_t}$, $t = 1, \cdots, N$ such that $\big(\boldsymbol{w}^*, \{\Pi^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}\big)$ satisfies the KKT system (12). Thus, the solution set of the KKT system (12) is also nonempty. This proves statement (i).

Statement (ii). Let $\big(\boldsymbol{w}^*, \{\Pi^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}\big)$ be a solution of the KKT system (12). It follows from statement (i) that such a solution exists. Now, consider $\boldsymbol{u}^* = \sum_{t=1}^N \boldsymbol{y}^{(t),*}$, $\boldsymbol{\lambda}^* = \boldsymbol{w}^*$, $\Lambda^{(t),*} = \Pi^{(t),*}$, $V^{(t),*} = D^{(t)} + \boldsymbol{y}^{(t),*}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t),*})^\top$, $t = 1, \cdots, N$. Then, by simple calculations and recalling (2), one can verify that $\big(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}, \boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ satisfies the KKT system (13). Hence, the solution set of the KKT system (13) is nonempty. Moreover, from (Ruszczyński, 2006, Theorem 3.27), we see that $\big(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}\big)$ is also an optimal solution of (9). This shows that the optimal solution of (9) exists.

Statement (iii). First, it is easy to see from (Ruszczyński, 2006, Theorem 3.27) that $(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ solves problem (9). Then, simplifying the KKT system (13) and recalling (2), one can verify that $(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ satisfies the KKT system (12) with $\boldsymbol{\lambda}^*$ in place of $\boldsymbol{w}$ and $\Lambda^{(t),*}$ in place of $\Pi^{(t)}$. Now, using (Ruszczyński, 2006, Theorem 3.27) again, we see that $\big(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ is an optimal solution of (6). This proves statement (iii). ■

In order to present the global convergence of Algorithm 1 based on the theory developed in (Fazel et al., 2013; Li et al., 2016), we first express problem (9) as follows:

$$\begin{array}{cl} \displaystyle\min_{\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}} & \theta\big(\boldsymbol{u}, \{V^{(t)}\}\big) + g\big(\{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big) \\ \text{s.t.} & A \begin{bmatrix} \boldsymbol{u} \\ \mathrm{vec}(V^{(1)}) \\ \vdots \\ \mathrm{vec}(V^{(N)}) \end{bmatrix} + B_1 \begin{bmatrix} \boldsymbol{y}^{(1)} \\ \vdots \\ \boldsymbol{y}^{(N)} \end{bmatrix} + B_2 \begin{bmatrix} \boldsymbol{z}^{(1)} \\ \vdots \\ \boldsymbol{z}^{(N)} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathrm{vec}(D^{(1)}) \\ \vdots \\ \mathrm{vec}(D^{(N)}) \end{bmatrix}, \end{array}$$

where $\theta\big(\boldsymbol{u}, \{V^{(t)}\}\big) = \delta^*_{\Delta_m}(\boldsymbol{u}) + \sum_{t=1}^N \delta^t_+(V^{(t)})$, $g\big(\{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big) = \sum_{t=1}^N \langle \boldsymbol{z}^{(t)}, \boldsymbol{a}^{(t)} \rangle$ and

$$A = \begin{bmatrix} -I_m & \\ & I_{m\sum_t m_t} \end{bmatrix}, \quad B_1 = \begin{bmatrix} 1 & \cdots & 1 \\ -\boldsymbol{e}_{m_1} & & \\ & \ddots & \vdots \\ & & -\boldsymbol{e}_{m_N} \end{bmatrix} \otimes I_m, \quad B_2 = \begin{bmatrix} 0 & \cdots & 0 \\ -I_{m_1} & & \\ & \ddots & \vdots \\ & & -I_{m_N} \end{bmatrix} \otimes \boldsymbol{e}_m. \tag{14}$$

It is easy to verify that $A^\top A = I_{m(1+\sum_t m_t)} \succ 0$ and

$$B_1^\top B_1 = \begin{bmatrix} m_1+1 & & \\ & \ddots & \\ & & m_N+1 \end{bmatrix} \otimes I_m \succ 0, \quad B_2^\top B_2 = m \begin{bmatrix} I_{m_1} & & \\ & \ddots & \\ & & I_{m_N} \end{bmatrix} \succ 0.$$

For notational simplicity, denote

$$\begin{aligned}
&\mathcal{W} := (\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}, \boldsymbol{\lambda}, \{\Lambda^{(t)}\}),\\
&\mathcal{W}^k := (\boldsymbol{u}^k, \{V^{(t),k}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}, \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}),\\
&\boldsymbol{y} := [\boldsymbol{y}^{(1)}; \cdots; \boldsymbol{y}^{(N)}], \quad \boldsymbol{y}^k := [\boldsymbol{y}^{(1),k}; \cdots; \boldsymbol{y}^{(N),k}],\\
&\boldsymbol{z} := [\boldsymbol{z}^{(1)}; \cdots; \boldsymbol{z}^{(N)}], \quad \boldsymbol{z}^k := [\boldsymbol{z}^{(1),k}; \cdots; \boldsymbol{z}^{(N),k}],\\
&\boldsymbol{v} := [\mathrm{vec}(V^{(1)}); \cdots; \mathrm{vec}(V^{(N)})], \quad \boldsymbol{v}^k := [\mathrm{vec}(V^{(1),k}); \cdots; \mathrm{vec}(V^{(N),k})],\\
&\boldsymbol{d} := [0; \mathrm{vec}(D^{(1)}); \cdots; \mathrm{vec}(D^{(N)})], \quad \mathrm{vec}(\{\Lambda^{(t)}\}) := [\mathrm{vec}(\Lambda^{(1)}); \cdots; \mathrm{vec}(\Lambda^{(N)})],\\
&\mathrm{vec}(\mathcal{W}) := [\boldsymbol{u}; \boldsymbol{v}; \boldsymbol{y}; \boldsymbol{z}; \boldsymbol{\lambda}; \mathrm{vec}(\{\Lambda^{(t)}\})].
\end{aligned}$$

By using the above notation, we can rewrite problem (9) in a compact form as follows:

$$\begin{aligned} \min \quad & \theta(\boldsymbol{u}, \boldsymbol{v}) + g(\boldsymbol{y}, \boldsymbol{z}) \\ \text{s.t.} \quad & A[\boldsymbol{u}; \boldsymbol{v}] + B[\boldsymbol{y}; \boldsymbol{z}] = \boldsymbol{d}, \end{aligned} \tag{15}$$

where $B = [B_1\ B_2]$. Then, our sGS-ADMM (Algorithm 1) is precisely a 2-block sPADMM applied to the compact form (15) of (9) with a specially designed proximal term. In particular, **Step 1** of the algorithm is the same as computing

$$(\boldsymbol{u}^{k+1}, \boldsymbol{v}^{k+1}) = \arg\min_{\boldsymbol{u},\boldsymbol{v}} \left\{ \mathcal{L}_\beta(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y}^k, \boldsymbol{z}^k; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}) \right\}. \tag{16}$$

It follows from (Li et al., 2016, Proposition 5) that **Step 2a–2c** is equivalent to

$$(\boldsymbol{y}^{k+1}, \boldsymbol{z}^{k+1}) = \arg\min_{\boldsymbol{y},\boldsymbol{z}} \left\{ \mathcal{L}_\beta(\boldsymbol{u}^{k+1}, \boldsymbol{v}^{k+1}, \boldsymbol{y}, \boldsymbol{z}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}) + \tfrac{\beta}{2}\|[\boldsymbol{y};\boldsymbol{z}] - [\boldsymbol{y}^k; \boldsymbol{z}^k]\|_C^2 \right\}, \tag{17}$$

where the matrix C in the proximal term is the symmetric Gauss-Seidel decomposition operator of $B^\top B$ and it is given by

$$C = \begin{bmatrix} B_1^\top B_2 \left(B_2^\top B_2\right)^{-1} B_2^\top B_1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Based on the above fact that the sGS-ADMM can be reformulated as a 2-block sPADMM with a specially designed semi-proximal term, one can directly obtain the global convergence of Algorithm 1 from that of the 2-block sPADMM.

Theorem 6 *Let $\beta > 0$, $\tau \in (0, \frac{1+\sqrt{5}}{2})$ and $\left\{ \left(\boldsymbol{u}^k, \{V^{(t),k}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}, \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\} \right) \right\}$ be the sequence generated by the sGS-ADMM in Algorithm 1. Then, the sequence $\left\{ \left(\boldsymbol{u}^k, \{V^{(t),k}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\} \right) \right\}$ converges to an optimal solution of* (9) *and the sequence $\left\{ \left(\boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\} \right) \right\}$ converges to an optimal solution of* (6).

Proof Here we apply the convergence result developed in (Fazel et al., 2013) to the 2-block sPADMM outlined in (16), (17) and **Step 3** of Algorithm 1. Since both $A^\top A$ and $\beta C + \beta B^\top B$ are positive definite, the conditions for ensuring the convergence of the 2-block sPADMM in (Fazel et al., 2013, Theorem B.1) are satisfied, thus along with Proposition 5, one can readily apply (Fazel et al., 2013, Theorem B.1) to obtain the desired results. ■

Moreover, based on the equivalence of our sGS-ADMM to a 2-block sPADMM, the linear convergence rate of the sGS-ADMM can also be established from the linear convergence result of the 2-block sPADMM; see (Han et al., 2018, Section 4.1) for more details.

Define

$$M := \begin{bmatrix} 0 & & \\ & \beta C + \beta B^\top B & \\ & & (\tau\beta)^{-1} I_{m\left(1+\sum_{t=1}^N m_t\right)} \end{bmatrix} + s_\tau \beta \begin{bmatrix} A^\top A & A^\top B & 0 \\ B^\top A & B^\top B & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

where A, B_1, B_2 are defined in (14) and $s_\tau := (5 - \tau - 3\min\{\tau,\, \tau^{-1}\})/4$. One can verify that $M \succ 0$. Indeed, it is easy to see from the definition that $M \succ 0$ if and only if

$$M_1 := \begin{bmatrix} A^\top A & A^\top B \\ B^\top A & s_\tau^{-1} C + (1 + s_\tau^{-1}) B^\top B \end{bmatrix} \succ 0.$$

Thus, one only needs to verify that $M_1 \succ 0$. Note that $A^\top A = AA^\top = I_{m\left(1+\sum_{t=1}^N m_t\right)} \succ 0$. The Schur complement of $A^\top A$ takes the form of

$$\begin{aligned} M_2 &:= s_\tau^{-1} C + (1 + s_\tau^{-1}) B^\top B - B^\top A (A^\top A)^{-1} A^\top B \;=\; s_\tau^{-1} C + s_\tau^{-1} B^\top B \\ &= s_\tau^{-1} \begin{bmatrix} B_1^\top B_2 \left(B_2^\top B_2\right)^{-1} B_2^\top B_1 + B_1^\top B_1 & B_1^\top B_2 \\ B_2^\top B_1 & B_2^\top B_2 \end{bmatrix}. \end{aligned}$$

Since $B_2^\top B_2 \succ 0$ and its Schur complement satisfies

$$B_1^\top B_2 \left(B_2^\top B_2\right)^{-1} B_2^\top B_1 + B_1^\top B_1 - B_1^\top B_2 \left(B_2^\top B_2\right)^{-1} B_2^\top B_1 = B_1^\top B_1 \succ 0,$$

then $M_2 \succ 0$. This implies that $M_1 \succ 0$ and hence $M \succ 0$.

We also let $\mathscr{W} := \mathbb{R}^m \times \otimes_{t=1}^N \mathbb{R}^{m \times m_t} \times \mathbb{R}^m \times \otimes_{t=1}^N \mathbb{R}^{m_t} \times \mathbb{R}^m \times \otimes_{t=1}^N \mathbb{R}^{m \times m_t}$ and $\Omega \subseteq \mathscr{W}$ be the solution set of the KKT system (13). Recall from Proposition 5(ii) that Ω is nonempty. Moreover, for any $\mathcal{W} \in \mathscr{W}$, we define

$$\begin{aligned} \mathrm{dist}(\mathcal{W},\, \Omega) &:= \inf_{\mathcal{W}' \in \Omega} \|\mathrm{vec}(\mathcal{W}) - \mathrm{vec}(\mathcal{W}')\|, \\ \mathrm{dist}_M(\mathcal{W},\, \Omega) &:= \inf_{\mathcal{W}' \in \Omega} \|\mathrm{vec}(\mathcal{W}) - \mathrm{vec}(\mathcal{W}')\|_M. \end{aligned}$$

Since $M \succ 0$, dist_M is also a point-to-set distance. We present the linear convergence result of our sGS-ADMM in the next theorem.

Theorem 7 *Let $\beta > 0$, $\tau \in (0, \frac{1+\sqrt{5}}{2})$ and $\{\mathcal{W}^k\}$ be the sequence generated by the sGS-ADMM in Algorithm 1. Then, there exists a constant $0 < \rho < 1$ such that, for all $k \geq 1$,*

$$\mathrm{dist}_M^2(\mathcal{W}^{k+1},\, \Omega) + \beta \big\| [\boldsymbol{y}^{k+1}; \boldsymbol{z}^{k+1}] - [\boldsymbol{y}^k; \boldsymbol{z}^k] \big\|_C^2 \leq \rho \Big(\mathrm{dist}_M^2(\mathcal{W}^k,\, \Omega) + \beta \big\| [\boldsymbol{y}^k; \boldsymbol{z}^k] - [\boldsymbol{y}^{k-1}; \boldsymbol{z}^{k-1}] \big\|_C^2 \Big).$$

Proof First we note the equivalence of the sGS-ADMM to a 2-block sPADMM. Next consider the KKT mapping $\mathcal{R} : \mathscr{W} \to \mathscr{W}$ defined by

$$\mathcal{R}(\mathcal{W}) := \begin{pmatrix} \boldsymbol{\lambda} - \mathrm{Pr}_{\Delta_m}(\boldsymbol{\lambda} + \boldsymbol{u}) \\ \left\{ V^{(t)} - \mathrm{Pr}_{+}^{t}(V^{(t)} - \Lambda^{(t)}) \right\} \\ \left\{ \Lambda^{(t)} \boldsymbol{e}_{m_t} - \boldsymbol{\lambda} \right\} \\ \left\{ (\Lambda^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)} \right\} \\ \sum_{t=1}^{N} \boldsymbol{y}^{(t)} - \boldsymbol{u} \\ \left\{ V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)} \boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m (\boldsymbol{z}^{(t)})^\top \right\} \end{pmatrix}, \quad \forall \mathcal{W} \in \mathscr{W},$$

where $\mathrm{Pr}_{\Delta_m}(\cdot)$ denotes the projection operator over Δ_m and $\mathrm{Pr}_{+}^{t}(\cdot)$ denotes the projection operator over $\mathbb{R}_{+}^{m \times m_t}$ for $t = 1, \cdots, N$. It is easy to see that $\mathcal{R}(\cdot)$ is continuous on $\mathscr{W}$. Moreover, note that $\boldsymbol{\lambda} \in \partial \delta_{\Delta_m}^{*}(\boldsymbol{u}) \Longleftrightarrow \boldsymbol{u} \in \partial \delta_{\Delta_m}(\boldsymbol{\lambda}) \Longleftrightarrow 0 \in \partial \delta_{\Delta_m}(\boldsymbol{\lambda}) + \boldsymbol{\lambda} - (\boldsymbol{\lambda} + \boldsymbol{u}) \Longleftrightarrow \boldsymbol{\lambda} = \mathrm{Prox}_{\delta_{\Delta_m}}(\boldsymbol{\lambda} + \boldsymbol{u}) = \mathrm{Pr}_{\Delta_m}(\boldsymbol{\lambda} + \boldsymbol{u})$, where the first equivalence follows from (2). Similarly, $-\Lambda^{(t)} \in \partial \delta_{+}^{t}(V^{(t)}) \Longleftrightarrow V^{(t)} = (V^{(t)} - \Lambda^{(t)})_{+}$, where $(\cdot)_{+} = \max(\cdot, 0)$. Using these facts, one can easily see that $\mathcal{R}(\mathcal{W}) = 0$ if and only if $\mathcal{W} \in \Omega$. By Theorem 6, we know that the sequence $\{\mathcal{W}^k\}$ converges to an optimal solution $\mathcal{W}^* \in \Omega$, and hence $\mathcal{R}(\mathcal{W}^*) = 0$.

Now, since Δ_m, $\mathbb{R}_{+}^{m \times m_1}, \cdots, \mathbb{R}_{+}^{m \times m_N}$ are polyhedral, it follows from (Rockafellar and Wets, 1998, Example 11.18) and the definition of projections that $\mathrm{Pr}_{\Delta_m}(\cdot)$ and $\mathrm{Pr}_{+}^{t}(\cdot)$ are piecewise polyhedral. Hence, $\mathcal{R}(\cdot)$ is also piecewise polyhedral. From (Robinson, 1981), we know that the KKT mapping $\mathcal{R}$ satisfies the following error bound condition: there exist two positive scalars $\eta > 0$ and $\tilde{\rho} > 0$ such that

$$\mathrm{dist}(\mathcal{W}, \Omega) \;\leq\; \eta \|\mathrm{vec}(\mathcal{R}(\mathcal{W}))\|, \quad \forall\, \mathcal{W} \in \{\mathcal{W} \mid \|\mathrm{vec}(\mathcal{R}(\mathcal{W}))\| \leq \tilde{\rho}\},$$

where $\mathrm{vec}(\mathcal{R}(\mathcal{W}))$ denotes the vectorization of $\mathcal{R}(\mathcal{W})$.

Finally, based on the above facts and Proposition 5, we can apply (Han et al., 2018, Corollary 1) to obtain the desired results. ■

5. Numerical Experiments

In this section, we conduct numerical experiments to test our sGS-ADMM in Algorithm 1 for computing Wasserstein barycenters with pre-specified support points, i.e., solving problem (4). In all our experiments, we use the 2-Wasserstein distance. We also compare our sGS-ADMM with the commercial software Gurobi and two existing representative methods, namely, the iterative Bregman projection (IBP) method (Benamou et al., 2015) and the modified Bregman ADMM (BADMM) (Ye et al., 2017). For ease of future reference, we briefly recall IBP and BADMM in Appendices A and B, respectively. All experiments are run in Matlab R2016a on a workstation with Intel(R) Xeon(R) Processor E-2176G@3.70GHz (this processor has 6 cores and 12 threads) and 64GB of RAM, equipped with 64-bit Windows 10 OS.

5.1 Implementation Details

In our implementation of the sGS-ADMM, a data scaling technique is used. Let $\kappa = \|[D^{(1)}, \cdots, D^{(N)}]\|_F$. Then, problem (4) is equivalent to

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\,\{\Pi^{(t)}\}} & \sum_{t=1}^{N} \langle \widehat{D}^{(t)},\, \Pi^{(t)} \rangle \\ \text{s.t.} & \Pi^{(t)} \boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0,\quad \forall\, t = 1, \cdots, N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{18}$$

where $\widehat{D}^{(t)} = \kappa^{-1} D^{(t)}$ for $t = 1, \cdots, N$. We then apply the sGS-ADMM to solve the dual problem of (18) to obtain an optimal solution of (4). Indeed, this technique has been widely used in ADMM-based methods to improve their numerical performances; see, for example, (Lam et al., 2018). Its effectiveness has also been observed in our experiments.

For a set of vectors $\{\boldsymbol{a}^{(t)} \,|\, t = 1,\cdots, N\}$, we define the notation $\|\{\boldsymbol{a}^{(t)}\}\| := \big(\sum_{t=1}^{N} \|\boldsymbol{a}^{(t)}\|^2\big)^{\frac{1}{2}}$. Similarly, for a set of matrices $\{A^{(t)} \mid t = 1, \ldots, N\}$, we define the notation $\|\{A^{(t)}\}\|_F := \big(\sum_{t=1}^{N} \|A^{(t)}\|_F^2\big)^{\frac{1}{2}}$. For any $\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}, \boldsymbol{\lambda}, \{\Lambda^{(t)}\}$, we define the relative residuals based on the KKT system (13) as follows:

$$\begin{array}{ll} \eta_1(\boldsymbol{\lambda}, \boldsymbol{u}) = \frac{\|\boldsymbol{\lambda} - \mathrm{Pr}_{\Delta_m}(\boldsymbol{\lambda} + \boldsymbol{u})\|}{1 + \|\boldsymbol{\lambda}\| + \|\boldsymbol{u}\|}, & \eta_2(\{V^{(t)}\}, \{\Lambda^{(t)}\}) = \frac{\|\{V^{(t)} - (V^{(t)} - \Lambda^{(t)})_+\}\|_F}{1 + \|\{V^{(t)}\}\|_F + \|\{\Lambda^{(t)}\}\|_F}, \\ \eta_3(\boldsymbol{\lambda}, \{\Lambda^{(t)}\}) = \frac{\|\{\Lambda^{(t)} \boldsymbol{e}_{m_t} - \boldsymbol{\lambda}\}\|}{1 + \|\boldsymbol{\lambda}\| + \|\{\Lambda^{(t)}\}\|_F}, & \eta_4(\{\Lambda^{(t)}\}) = \frac{\|\{(\Lambda^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)}\}\|_F}{1 + \|\{\boldsymbol{a}^{(t)}\}\| + \|\{\Lambda^{(t)}\}\|_F}, \\ \eta_5(\boldsymbol{u}, \{\boldsymbol{y}^{(t)}\}) = \frac{\|\sum_{t=1}^{N} \boldsymbol{y}^{(t)} - \boldsymbol{u}\|}{1 + \|\sum_{t=1}^{N} \boldsymbol{y}^{(t)}\| + \|\boldsymbol{u}\|}, & \eta_6(\{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}) = \frac{\|\{V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)} \boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m (\boldsymbol{z}^{(t)})^\top\}\|_F}{1 + \|\{D^{(t)}\}\|_F + \|\{V^{(t)}\}\|_F + \|\{\boldsymbol{y}^{(t)}\}\| + \|\{\boldsymbol{z}^{(t)}\}\|}, \\ \eta_7(\boldsymbol{\lambda}) = \frac{|\boldsymbol{e}_m^\top \boldsymbol{\lambda} - 1| + \|\min(\boldsymbol{\lambda}, 0)\|}{1 + \|\boldsymbol{\lambda}\|}, & \eta_8(\{\Lambda^{(t)}\}) = \frac{\|\min([\Lambda^{(1)}, \cdots, \Lambda^{(N)}], 0)\|_F}{1 + \|\{\Lambda^{(t)}\}\|_F}. \end{array}$$

Moreover, let $\mathcal{W} = (\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}, \boldsymbol{\lambda}, \{\Lambda^{(t)}\})$ and

$$\begin{aligned} \eta_P(\mathcal{W}) &= \max\big\{\eta_1(\boldsymbol{\lambda}, \boldsymbol{u}),\, 0.7\eta_2\big(\{V^{(t)}\}, \{\Lambda^{(t)}\}\big),\, \eta_3\big(\boldsymbol{\lambda}, \{\Lambda^{(t)}\}\big),\, \eta_4\big(\{\Lambda^{(t)}\}\big)\big\}, \\ \eta_D(\mathcal{W}) &= \max\big\{0.7\eta_5\big(\boldsymbol{u}, \{\boldsymbol{y}^{(t)}\}\big),\, \eta_6\big(\{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big),\, \eta_7(\boldsymbol{\lambda}),\, 0.7\eta_8\big(\{\Lambda^{(t)}\}\big)\big\}. \end{aligned}$$

Following discussions in Theorem 7, it is easy to verify that $\max\{\eta_P(\mathcal{W}), \eta_D(\mathcal{W})\} = 0$ if and only if $\mathcal{W}$ is a solution of the KKT system (13). The relative duality gap is defined by

$$\eta_{gap}(\mathcal{W}) := \frac{|\,\mathrm{obj}_P(\mathcal{W}) - \mathrm{obj}_D(\mathcal{W})\,|}{1 + |\,\mathrm{obj}_P(\mathcal{W})\,| + |\,\mathrm{obj}_D(\mathcal{W})\,|},$$

where $\mathrm{obj}_P(\mathcal{W}) = \sum_{t=1}^{N} \langle D^{(t)},\, \Pi^{(t)} \rangle$ and $\mathrm{obj}_D(\mathcal{W}) = \delta^*_{\Delta_m}\big(\sum_{t=1}^{N} \boldsymbol{y}^{(t)}\big) + \sum_{t=1}^{N} \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)} \rangle$. We use these relative residuals in our stopping criterion for the sGS-ADMM. Specifically, we will terminate the sGS-ADMM when

$$\max\big\{\eta_P(\mathcal{W}^{k+1}),\, \eta_D(\mathcal{W}^{k+1}),\, \eta_{gap}(\mathcal{W}^{k+1})\big\} < \mathrm{Tol}_{\mathrm{sgs}},$$

where $\mathcal{W}^{k+1}$ is generated by the sGS-ADMM at the k-th iteration and the value of $\mathrm{Tol}_{\mathrm{sgs}}$ will be given later.

We also use a similar numerical strategy as (Lam et al., 2018, Section 4.4) to update the penalty parameter β in the augmented Lagrangian function at every 50 iterations. Specifically, set $\beta_0 = 1$. At the k-th iteration, if $\mathrm{mod}(k,\,50) \neq 0$, set $\beta_{k+1} = \beta_k$; otherwise, compute $\chi^{k+1} = \frac{\eta_D(\mathcal{W}^{k+1})}{\eta_P(\mathcal{W}^{k+1})}$ and then, set

$$\beta_{k+1} = \begin{cases} \sigma\beta_k, & \text{if } \ \chi^{k+1} > 2, \\ \sigma^{-1}\beta_k, & \text{if } \ \frac{1}{\chi^{k+1}} > 2, \\ \beta_k, & \text{otherwise,} \end{cases} \quad \text{with } \ \sigma = \begin{cases} 1.1, & \text{if } \ \max\{\chi^{k+1},\, \frac{1}{\chi^{k+1}}\} \leq 50, \\ 2, & \text{if } \ \max\{\chi^{k+1},\, \frac{1}{\chi^{k+1}}\} > 500, \\ 1.5, & \text{otherwise,} \end{cases}$$

where $\mathrm{mod}(k,\,50)$ denotes the remainder after division of k by 50. Note that the value of β is adjusted based on the primal and dual information. As observed from our experiments, this updating strategy can efficiently balance the convergence of the primal and dual variables, and improve the convergence speed of our algorithm.

Computing all the above residuals is expensive. Thus, in our implementations, we only compute them and check the termination criteria at every 50 iterations. In addition, we initialize the sGS-ADMM at origin and choose the dual step-size τ to be 1.618.

For IBP, the regularization parameter ε is chosen from $\{0.1, 0.01, 0.001\}$ in our experiments. For $\varepsilon \in \{0.1, 0.01\}$, we follow (Benamou et al., 2015, Remark 3) to implement the algorithm (see (20)) and terminate it when

$$\frac{\|\boldsymbol{w}^{k+1}-\boldsymbol{w}^k\|}{1+\|\boldsymbol{w}^{k+1}\|+\|\boldsymbol{w}^k\|} < \mathrm{Tol_{ibp}}, \ \frac{\|\{\boldsymbol{u}^{(t),k+1}-\boldsymbol{u}^{(t),k}\}\|}{1+\|\{\boldsymbol{u}^{(t),k+1}\}\|+\|\{\boldsymbol{u}^{(t),k}\}\|} < \mathrm{Tol_{ibp}}, \ \frac{\|\{\boldsymbol{v}^{(t),k+1}-\boldsymbol{v}^{(t),k}\}\|_F}{1+\|\{\boldsymbol{v}^{(t),k+1}\}\|+\|\{\boldsymbol{v}^{(t),k}\}\|} < \mathrm{Tol_{ibp}},$$

where $(\boldsymbol{w}^{k+1}, \{\boldsymbol{u}^{(t),k+1}\}, \{\boldsymbol{v}^{(t),k+1}\})$ is generated at the k-th iteration in (20). Moreover, for $\varepsilon = 0.001$, we follow (Peyré and Cuturi, 2019, Section 4.4) to adapt the *log-sum-exp* trick for stabilizing IBP (see (21)). This stabilized IBP is terminated when

$$\frac{\|\tilde{\boldsymbol{w}}^{k+1}-\tilde{\boldsymbol{w}}^k\|}{1+\|\tilde{\boldsymbol{w}}^{k+1}\|+\|\tilde{\boldsymbol{w}}^k\|} < \mathrm{Tol_{ibp}}, \ \frac{\|\{\tilde{\boldsymbol{u}}^{(t),k+1}-\tilde{\boldsymbol{u}}^{(t),k}\}\|}{1+\|\{\tilde{\boldsymbol{u}}^{(t),k+1}\}\|+\|\{\tilde{\boldsymbol{u}}^{(t),k}\}\|} < \mathrm{Tol_{ibp}}, \ \frac{\|\{\tilde{\boldsymbol{v}}^{(t),k+1}-\tilde{\boldsymbol{v}}^{(t),k}\}\|_F}{1+\|\{\tilde{\boldsymbol{v}}^{(t),k+1}\}\|+\|\{\tilde{\boldsymbol{v}}^{(t),k}\}\|} < \mathrm{Tol_{ibp}},$$

where $(\tilde{\boldsymbol{w}}^{k+1}, \{\tilde{\boldsymbol{u}}^{(t),k+1}\}, \{\tilde{\boldsymbol{v}}^{(t),k+1}\})$ is generated at the k-th iteration in (21). The value of $\mathrm{Tol_{ibp}}$ will be given later.

For BADMM, we use the Matlab codes[2] implemented by the authors in (Ye et al., 2017) and terminate them when

$$\begin{aligned} &\max\big\{\eta_3\big(\boldsymbol{w}^{k+1}, \{\Gamma^{(t),k+1}\}\big),\, \eta_4(\{\Pi^{(t),k+1}\})\big\} < \mathrm{Tol_b}, && \frac{\|\boldsymbol{w}^{k+1}-\boldsymbol{w}^k\|}{1+\|\boldsymbol{w}^{k+1}\|+\|\boldsymbol{w}^k\|} < \mathrm{Tol_b}, \\ &\frac{\|\{\Pi^{(t),k+1}-\Gamma^{(t),k+1}\}\|_F}{1+\|\{\Pi^{(t),k+1}\}\|_F+\|\{\Gamma^{(t),k+1}\}\|_F} < \mathrm{Tol_b}, && \frac{\|\{\Pi^{(t),k+1}-\Pi^{(t),k}\}\|_F}{1+\|\{\Pi^{(t),k}\}\|_F+\|\{\Pi^{(t),k+1}\}\|_F} < \mathrm{Tol_b}, \\ &\frac{\|\{\Gamma^{(t),k+1}-\Gamma^{(t),k}\}\|_F}{1+\|\{\Gamma^{(t),k}\}\|_F+\|\{\Gamma^{(t),k+1}\}\|_F} < \mathrm{Tol_b}, && \frac{\|\{\Lambda^{(t),k+1}-\Lambda^{(t),k}\}\|_F}{1+\|\{\Lambda^{(t),k}\}\|_F+\|\{\Lambda^{(t),k+1}\}\|_F} < \mathrm{Tol_b}, \end{aligned}$$

where $(\boldsymbol{w}^{k+1}, \{\Pi^{(t),k+1}\}, \{\Gamma^{(t),k+1}\}, \{\Lambda^{(t),k+1}\})$ is generated by BADMM at the k-th iteration (see Appendix B) and the value of $\mathrm{Tol_b}$ will be given later. The above termination criteria are checked at every 200 iterations.

We also apply Gurobi 8.0.0 (Gurobi Optimization, 2018) to solve problem (4). It is well known that Gurobi is a highly powerful commercial package for solving linear programming

2. Available in `https://github.com/bobye/WBC_Matlab`.

problems and can provide high quality solutions. Therefore, we use the solution obtained by Gurobi as a benchmark to evaluate the qualities of solutions obtained by different methods. In our experiments, we use the default parameter settings for Gurobi. Note that, by the default settings, Gurobi actually uses a concurrent optimization strategy to solve LPs, which runs multiple classical LP solvers (the primal/dual simplex method and the barrier method) on multiple threads simultaneously and chooses the one that finishes first.

We shall conduct the experiments as follows. In subsection 5.2, we test different methods on synthetic data to show their computational performance in terms of accuracy and speed. In subsection 5.3, we test on the MNIST data set to visualize the quality of results obtained by each method. In subsection 5.4, we conduct some experiments for the free support case. A summary of our experiments is given in subsection 5.5.

5.2 Experiments on Synthetic Data

In this subsection, we generate a set of discrete probability distributions $\{\mathcal{P}^{(t)}\}_{t=1}^N$ with $\mathcal{P}^{(t)} = \big\{(a_i^{(t)},\, \boldsymbol{q}_i^{(t)}) \in \mathbb{R}_+ \times \mathbb{R}^d : i = 1, \cdots, m_t\big\}$ and $\sum_{i=1}^{m_t} a_i^{(t)} = 1$, and then apply different methods to solve problem (4) to compute a Wasserstein barycenter $\mathcal{P} = \big\{(w_i,\, \boldsymbol{x}_i) \in \mathbb{R}_+ \times \mathbb{R}^d : i = 1, \cdots, m\big\}$, where m and $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m)$ are pre-specified. Specifically, we set $d = 3$, $\gamma_1 = \cdots = \gamma_N = \frac{1}{N}$ and $m_1 = \cdots = m_N = m'$ for convenience, and choose different (N, m, m'). Then, given each triple (N, m, m'), we randomly generate a trial in the following three cases.

- **Case 1. Each distribution has different *dense* weights (all weights are nonzero) and different support points.** In this case, we first generate the support points $\{\boldsymbol{q}_i^{(t)} : i = 1, \cdots, m',\, t = 1, \cdots, N\}$ whose entries are drawn from a Gaussian mixture distribution via the following Matlab commands:

```
gm_num = 5; gm_mean = [-20; -10; 0; 10; 20];
sigma = zeros(1,1,gm_num); sigma(1,1,:) = 5*ones(gm_num,1);
gm_weights = rand(gm_num,1);
distrib = gmdistribution(gm_mean, sigma, gm_weights);
```

  Next, for each t, we generate an associated weight vector $(a_1^{(t)}, \cdots, a_{m'}^{(t)})$ whose entries are drawn from the standard uniform distribution on the open interval $(0, 1)$, and then normalize it so that $\sum_{i=1}^{m'} a_i^{(t)} = 1$. After generating all $\{\mathcal{P}^{(t)}\}_{t=1}^N$, we use the k-means[3] method to choose m points from $\{\boldsymbol{q}_i^{(t)} : i = 1, \cdots, m',\, t = 1, \cdots, N\}$ to be the support points of the barycenter.

- **Case 2. Each distribution has different *sparse* weights (most of weights are zeros) and different support points.** In this case, we also generate the support points $\{\boldsymbol{q}_i^{(t)} : i = 1, \cdots, m',\, t = 1, \cdots, N\}$ whose entries are drawn from a Gaussian mixture distribution as in **Case 1**. Next, for each t, we choose a subset $\mathcal{S}_t \subset \{1, \cdots, m'\}$ of size s uniformly at random and generate an s-sparse weight vector $(a_1^{(t)}, \cdots, a_{m'}^{(t)})$, which has uniformly distributed entries in the interval $(0, 1)$ on $\mathcal{S}_t$ and

3. In our experiments, we call the Matlab function "`kmeans`", which is built in statistics and machine learning toolbox.

zeros on $\mathcal{S}_t^c$. Then, we normalize it so that $\sum_{i=1}^{m'} a_i^{(t)} = 1$. The number s is set to be $\lfloor m' \times \texttt{sr} \rfloor$, where `sr` denotes the sparsity ratio and $\lfloor a \rfloor$ denotes the greatest integer less than or equal to a. The number m is set to be larger than s. The support points of the barycenter are chosen from $\{\boldsymbol{q}_i^{(t)} : a_i^{(t)} \neq 0,\, i = 1, \cdots, m',\, t = 1, \cdots, N\}$ by the k-means method. Note that, in this case, one can solve a smaller problem (5) to obtain an optimal solution of (4); see Remark 1.

- **Case 3. Each distribution has different *dense* weights (all weights are nonzero), but has the same support points.** In this case, we set $m = m'$ and generate the points $(\boldsymbol{q}_1, \cdots, \boldsymbol{q}_m)$ whose entries are drawn from a Gaussian mixture distribution as in **Case 1**. Then, all distributions $\{\mathcal{P}^{(t)}\}_{t=1}^N$ and the barycenter use $(\boldsymbol{q}_1, \cdots, \boldsymbol{q}_m)$ as the support points. Next, for each t, we generate an associated weight vector $(a_1^{(t)}, \cdots, a_m^{(t)})$ whose entries are drawn from the standard uniform distribution on the open interval $(0, 1)$, and then normalize it so that $\sum_{i=1}^m a_i^{(t)} = 1$.

Tables $1, 2, 3$ present numerical results of different methods for **Cases 1, 2, 3**, respectively, where we use different choices of $(N,\, m,\, m')$ and different sparsity ratio `sr`. In this part of experiments, we set $\mathrm{Tol}_{\mathrm{sgs}} = \mathrm{Tol}_{\mathrm{b}} = 10^{-5}$ and $\mathrm{Tol}_{\mathrm{ibp}} = 10^{-8}$ for termination. We also set the maximum numbers of iterations for sGS-ADMM, BADMM and IBP to 3000, 3000, 10000, respectively. In each table, "normalized obj" denotes the normalized objective value defined by $\frac{\left|\mathcal{F}(\{\Pi^{(t),*}\}) - \mathcal{F}_{\mathrm{gu}}\right|}{\mathcal{F}_{\mathrm{gu}}}$, where $\mathcal{F}(\{\Pi^{(t),*}\}) := \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t),*} \rangle$ with $(\boldsymbol{w}^*,\, \{\Pi^{(t),*}\})$ being the terminating solution obtained by each algorithm and $\mathcal{F}_{\mathrm{gu}}$ denotes the objective value obtained by Gurobi; "feasibility" denotes the value of

$$\eta_{\mathrm{feas}}\big(\boldsymbol{w}^*, \{\Pi^{(t),*}\}\big) := \max\big\{\eta_3\big(\boldsymbol{w}^*, \{\Pi^{(t),*}\}\big),\, \eta_4\big(\{\Pi^{(t),*}\}\big),\, \eta_7(\boldsymbol{w}^*),\, \eta_8\big(\{\Pi^{(t),*}\}\big)\big\},$$

which is used to measure the deviation of the terminating solution from the feasible set; "time" denotes the computational time (in seconds); "iter" denotes the number of iterations. All results presented are the average of 10 independent trials.

One can observe from Tables $1, 2, 3$ that our sGS-ADMM performs much better than BADMM and IBP ($\varepsilon = 0.001$) in the sense that it always returns an objective value considerably closer to that of Gurobi while achieving comparable feasibility accuracy in less computational time. For IBP with $\varepsilon \in \{0.1, 0.01\}$, it always converges faster and achieves better feasibility accuracy, but it gives a rather poor objective value, which means that the solution obtained is rather crude. Although a small $\varepsilon = 0.001$ can give a better approximation, it may also lead to the numerical instability. The *log-sum-exp* stabilization trick can be used to ameliorate this issue. However, with this trick, IBP (see (21)) must give up some computational efficiency in matrix-vector multiplications and require many additional exponential evaluations that are typically time-consuming. Moreover, when ε is small, the convergence of IBP becomes quite slow, as evident in three tables. For BADMM, it can give an objective value close to that of Gurobi. However, it takes much more time and its feasibility accuracy is the worst for most cases. Thus, the performance of BADMM is still not good enough. Moreover, the convergence of BADMM is still unknown. For Gurobi, when N, m and m' are relatively small, it can solve the problem highly efficiently. However, when the problem size becomes larger, Gurobi would take much more time. As an example, for the case where $(N,\, m,\, m') = (100,\, 300,\, 200)$ in Table 1, one would need to solve a large-scale LP containing 6000300 nonnegative variables and 50001 equality constraints. In this case, we see that Gurobi is about 20 times slower than our sGS-ADMM.

Table 1: Numerical results on synthetic data for **Case 1**. In this case, each distribution has different *dense* weights and different support points. In the table, "sGS" stands for sGS-ADMM; "BA" stands for BADMM; "IBP1" stands for IBP with $\varepsilon = 0.1$; "IBP2" stands for IBP with $\varepsilon = 0.01$; "IBP3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	sGS	BA	IBP1	IBP2	IBP3	Gurobi	sGS	BA	IBP1	IBP2	IBP3
			normalized obj					feasibility					
20	100	100	1.17e-4	5.54e-5	1.17e+0	7.09e-2	3.95e-2	1.05e-15	1.40e-5	2.00e-4	3.97e-9	3.92e-8	1.22e-4
20	200	100	2.45e-4	1.18e-4	1.30e+0	9.98e-2	6.60e-2	9.60e-16	1.39e-5	2.61e-4	2.68e-9	2.08e-8	3.91e-5
20	200	200	4.01e-4	1.05e-3	2.21e+0	1.28e-1	4.70e-2	2.41e-7	1.39e-5	3.07e-4	3.66e-9	2.63e-8	4.29e-5
20	300	200	4.65e-4	1.53e-3	2.33e+0	1.56e-1	6.61e-2	1.97e-7	1.41e-5	3.67e-4	2.66e-9	1.08e-8	1.45e-5
50	100	100	9.85e-5	1.20e-4	1.14e+0	6.40e-2	3.46e-2	2.03e-7	1.40e-5	2.92e-4	7.61e-9	1.30e-7	1.76e-4
50	200	100	1.57e-4	1.30e-4	1.24e+0	8.93e-2	5.76e-2	1.25e-7	1.41e-5	3.99e-4	5.83e-9	8.13e-8	1.01e-4
50	200	200	2.52e-4	1.29e-3	2.09e+0	1.20e-1	4.22e-2	1.76e-7	1.41e-5	4.60e-4	4.73e-9	3.63e-8	7.31e-5
50	300	200	4.02e-4	1.93e-3	2.21e+0	1.41e-1	5.74e-2	4.34e-7	1.40e-5	5.58e-4	3.81e-9	3.89e-8	3.07e-5
100	100	100	2.12e-4	1.35e-4	1.11e+0	6.24e-2	3.39e-2	2.48e-7	1.45e-5	3.63e-4	7.56e-9	9.03e-8	2.55e-4
100	200	100	3.32e-4	1.99e-4	1.21e+0	8.65e-2	5.68e-2	1.89e-7	1.43e-5	5.10e-4	6.16e-9	5.23e-8	1.08e-4
100	200	200	5.15e-4	1.35e-3	2.11e+0	1.21e-1	4.35e-2	3.42e-7	1.51e-5	5.89e-4	6.12e-9	7.69e-8	8.21e-5
100	300	200	6.56e-4	2.04e-3	2.21e+0	1.40e-1	5.53e-2	5.14e-7	1.47e-5	7.24e-4	5.00e-9	6.11e-8	3.88e-5
			iter					time (in seconds)					
20	100	100	2595	3000	112	2965	10000	1.84	3.23	33.27	0.14	3.45	24.91
20	200	100	2495	3000	107	1761	10000	6.67	8.48	69.20	0.25	3.85	47.51
20	200	200	2585	3000	103	2049	10000	10.56	19.24	139.12	0.50	9.39	98.31
20	300	200	2465	3000	102	1505	10000	23.68	28.14	208.91	0.76	10.59	152.04
50	100	100	2930	3000	112	4440	10000	9.21	13.18	85.33	0.33	12.30	60.53
50	200	100	2820	3000	110	2712	10000	53.21	27.36	175.70	0.68	15.94	127.48
50	200	200	2900	3000	104	2472	10000	72.73	56.66	341.90	1.30	29.45	250.09
50	300	200	2840	3000	103	1850	10000	299.94	85.01	517.10	1.95	33.35	376.42
100	100	100	2985	3000	117	5398	10000	9.89	28.92	173.16	0.74	32.72	127.03
100	200	100	2980	3000	110	2937	10000	31.03	58.46	347.86	1.40	35.72	254.26
100	200	200	3000	3000	105	2730	10000	63.72	117.03	690.55	2.61	64.81	503.19
100	300	200	3000	3000	102	1923	10000	3703.33	178.99	1032.84	3.80	68.31	756.29

We next follow (Cuturi and Peyré, 2016, Section 3.4) to conduct a simple example to visually show the qualities of the barycenter $\boldsymbol{w}^*$ and transport plans $\{\Pi^{(t),*}\}$ computed by different algorithms. Consider two one-dimensional continuous Gaussian distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$. It is known from (Agueh and Carlier, 2011, Section 6.2) and (McCann, 1997, Example 1.7) that their 2-Wasserstein barycenter is the Gaussian distribution $\mathcal{N}\left(\frac{\mu_1+\mu_2}{2}, \left(\frac{\sigma_1+\sigma_2}{2}\right)^2\right)$. Based on this fact, we discretize two Gaussian distributions $\mathcal{N}(-2, \left(\frac{1}{4}\right)^2)$ and $\mathcal{N}(2, 1)$, and then apply different algorithms to compute their barycenter, which is expected to be close to the discretization of the true barycenter $\mathcal{N}(0, \left(\frac{5}{8}\right)^2)$. The discretization is performed on the interval $[-4, 5]$ with n uniform grids. Since this part of experiments is not intended for comparing speed, we shall use tighter tolerances, say, $\mathrm{Tol}_{\mathrm{sgs}} = \mathrm{Tol}_{\mathrm{b}} = 10^{-6}$ and $\mathrm{Tol}_{\mathrm{ibp}} = 10^{-10}$, and set the maximum numbers of iterations for all algorithms to 20000. Figure 1(a) shows the barycenters computed by different algorithms for $n = 500$. From this figure, we see that the barycenter computed by Gurobi oscillates wildly. A similar result has also been observed in (Cuturi and Peyré, 2016, Section 3.4). The

Table 2: Numerical results on synthetic data for **Case 2**. In this case, each distribution has different *sparse* weights and different support points. In the table, "sGS" stands for sGS-ADMM; "BA" stands for BADMM; "IBP1" stands for IBP with $\varepsilon = 0.1$; "IBP2" stands for IBP with $\varepsilon = 0.01$; "IBP3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	sr	sGS	BA	IBP1	IBP2	IBP3	Gurobi	sGS	BA	IBP1	IBP2	IBP3
				normalized obj					feasibility					
50	50	500	0.1	4.22e-5	1.58e-4	5.52e-1	3.54e-2	2.60e-2	9.22e-16	1.45e-5	2.67e-4	1.67e-8	5.90e-7	5.46e-4
50	100	500	0.2	8.58e-5	1.38e-4	1.14e+0	6.19e-2	3.30e-2	9.28e-8	1.40e-5	2.92e-4	7.05e-9	6.29e-8	1.76e-4
50	100	1000	0.1	1.02e-4	1.51e-4	1.16e+0	6.47e-2	3.49e-2	8.89e-8	1.41e-5	2.76e-4	8.08e-9	7.59e-8	1.57e-4
50	200	1000	0.2	3.21e-4	1.30e-3	2.12e+0	1.22e-1	4.38e-2	8.11e-8	1.41e-5	4.65e-4	4.26e-9	3.69e-8	6.65e-5
100	50	500	0.1	6.26e-5	9.86e-5	5.62e-1	3.53e-2	2.42e-2	3.36e-8	1.49e-5	2.96e-4	2.00e-8	2.73e-7	6.47e-4
100	100	500	0.2	1.93e-4	1.68e-4	1.14e+0	6.08e-2	3.22e-2	2.36e-15	1.48e-5	3.65e-4	7.97e-9	8.39e-7	2.52e-4
100	100	1000	0.1	1.89e-4	1.56e-4	1.13e+0	6.07e-2	3.15e-2	1.79e-8	1.46e-5	3.62e-4	9.97e-9	8.65e-7	2.39e-4
100	200	1000	0.2	6.04e-4	1.29e-3	2.12e+0	1.22e-1	4.32e-2	3.19e-7	1.50e-5	5.84e-4	5.41e-9	7.17e-8	7.40e-5
200	50	500	0.1	1.31e-4	9.33e-5	5.63e-1	3.56e-2	2.38e-2	3.43e-8	1.51e-5	3.54e-4	3.54e-8	8.22e-7	7.21e-4
200	100	500	0.2	4.20e-4	1.61e-4	1.12e+0	6.01e-2	3.23e-2	1.06e-7	1.56e-5	4.39e-4	7.80e-9	2.50e-7	3.19e-4
200	100	1000	0.1	3.93e-4	1.65e-4	1.12e+0	6.16e-2	3.29e-2	1.97e-7	1.57e-5	4.35e-4	1.42e-8	3.25e-7	3.27e-4
200	200	1000	0.2	1.27e-3	1.35e-3	2.09e+0	1.20e-1	4.34e-2	3.09e-7	1.61e-5	7.25e-4	7.78e-9	2.31e-7	1.12e-4
				iter					time (in seconds)					
50	50	500	0.1	2850	3000	147	7790	10000	1.66	1.49	12.78	0.05	2.43	10.46
50	100	500	0.2	2965	3000	110	3098	10000	9.19	13.06	83.96	0.33	8.60	60.05
50	100	1000	0.1	2945	3000	109	4071	10000	9.13	12.98	84.11	0.32	11.29	59.90
50	200	1000	0.2	2885	3000	104	2294	10000	75.44	55.95	337.89	1.29	27.23	249.11
100	50	500	0.1	2965	3000	137	6915	10000	1.86	5.64	41.85	0.21	10.25	31.29
100	100	500	0.2	3000	3000	111	4520	10000	10.40	28.71	171.21	0.70	27.18	126.31
100	100	1000	0.1	3000	3000	118	5675	10000	11.01	28.76	171.31	0.74	34.22	126.46
100	200	1000	0.2	3000	3000	104	2985	10000	63.89	117.93	674.10	2.57	70.33	499.95
200	50	500	0.1	3000	3000	154	8143	10000	3.98	13.56	85.42	0.48	24.49	63.71
200	100	500	0.2	3000	3000	126	5600	10000	27.66	57.73	339.87	1.60	68.39	254.00
200	100	1000	0.1	3000	3000	116	5764	10000	31.36	57.75	340.34	1.47	70.34	254.03
200	200	1000	0.2	3000	3000	104	3107	10000	143.95	224.84	1366.46	5.14	146.97	1010.56

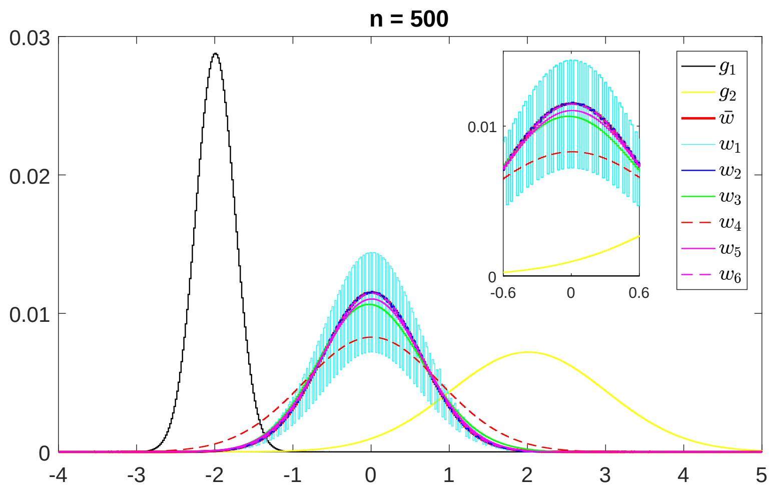
possible reason for this phenomenon is that the LP (4) has multiple solutions and Gurobi using a simplex method may not find a "smooth" one. IBP always finds a "smooth" solution thanks to the entropic regularization in the objective. A smaller ε (say, 0.001) indeed gives a better approximation. On the other hand, our sGS-ADMM and BADMM are also able to find a "smooth" barycenter, although they are designed to solve the original LP. This could be due to the fact that these two algorithms are developed based on the augmented Lagrangian function or its variants, and they implicitly have a 'smoothing' regularization (due to the penalty or proximal term) in each subproblem. In particular, just as IBP with $\varepsilon = 0.001$, the barycenter computed by the sGS-ADMM can match the true barycenter almost exactly. We also show the transport plans for $n = 500$ in Figure 1(b). One can see that the transport plans computed by sGS-ADMM are more similar to those computed by Gurobi, while the transport plans computed by IBP are more blurry. Consequently, these two figures clearly demonstrate the superior quality of the solution obtained by our sGS-ADMM.

To further compare the performances of Gurobi and our sGS-ADMM, we conduct more experiments on synthetic data for **Case 1**, where we fix two of three dimensions m, m',

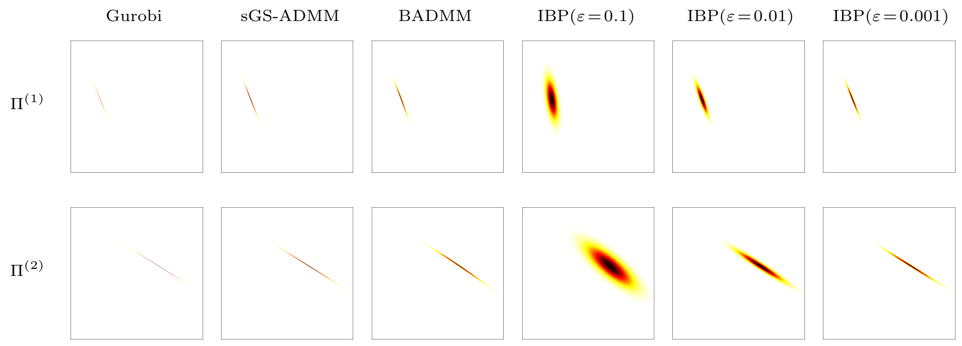
Table 3: Numerical results on synthetic data for **Case 3**. In this case, each distribution has different *dense* weights, but has the same support points. In the table, "sGS" stands for sGS-ADMM; "BA" stands for BADMM; "IBP1" stands for IBP with $\varepsilon = 0.1$; "IBP2" stands for IBP with $\varepsilon = 0.01$; "IBP3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	sGS	BA	IBP1	IBP2	IBP3	Gurobi	sGS	BA	IBP1	IBP2	IBP3
			normalized obj					feasibility					
20	50	50	1.68e-4	4.08e-4	1.02e+0	2.23e-2	4.48e-3	6.79e-16	1.42e-5	2.22e-4	1.28e-8	3.71e-6	1.17e-3
20	100	100	1.84e-4	4.12e-4	2.12e+0	6.26e-2	1.91e-3	1.97e-8	1.43e-5	2.92e-4	1.24e-8	1.49e-6	5.11e-4
20	200	200	8.12e-4	2.80e-3	4.34e+0	1.72e-1	1.59e-3	2.55e-7	1.40e-5	4.38e-4	4.10e-9	1.25e-6	2.49e-4
50	50	50	9.73e-5	6.26e-4	1.02e+0	2.18e-2	3.84e-3	4.11e-8	1.63e-5	3.18e-4	1.98e-8	1.18e-5	1.52e-3
50	100	100	2.47e-4	3.91e-4	2.07e+0	6.04e-2	2.48e-3	9.32e-8	1.64e-5	4.32e-4	1.25e-8	2.16e-6	8.39e-4
50	200	200	6.17e-4	2.81e-3	4.23e+0	1.65e-1	1.50e-3	3.37e-7	1.49e-5	6.46e-4	6.70e-9	1.37e-7	3.77e-4
100	50	50	1.39e-4	2.72e-4	1.02e+0	2.15e-2	3.95e-3	1.17e-7	1.85e-5	4.05e-4	3.16e-8	1.14e-5	1.95e-3
100	100	100	3.85e-4	4.13e-4	2.07e+0	6.00e-2	2.49e-3	1.88e-7	1.73e-5	5.27e-4	1.08e-8	4.97e-6	1.05e-3
100	200	200	1.06e-3	2.94e-3	4.19e+0	1.63e-1	1.46e-3	3.79e-7	1.65e-5	8.17e-4	6.30e-9	7.26e-7	4.91e-4
200	50	50	2.45e-4	2.87e-4	1.02e+0	2.15e-2	3.65e-3	5.21e-8	1.90e-5	4.43e-4	1.87e-7	1.31e-5	2.14e-3
200	100	100	7.75e-4	4.08e-4	2.05e+0	5.91e-2	2.59e-3	6.45e-8	1.81e-5	6.45e-4	1.87e-8	5.66e-6	1.28e-3
200	200	200	2.33e-3	2.96e-3	4.15e+0	1.61e-1	1.34e-3	3.43e-7	1.73e-5	1.02e-3	8.17e-9	9.32e-7	5.90e-4
			iter					time (in seconds)					
20	50	50	2895	3000	316	8465	10000	0.29	0.72	5.59	0.03	0.79	5.22
20	100	100	2925	3000	225	6383	10000	1.69	3.68	33.69	0.03	0.75	24.68
20	200	200	2765	3000	157	6037	10000	9.90	21.28	139.94	0.04	1.11	98.05
50	50	50	3000	3000	286	9815	10000	1.34	1.91	13.48	0.04	1.23	10.98
50	100	100	3000	3000	226	8759	10000	9.41	14.11	85.93	0.05	1.81	60.24
50	200	200	2995	3000	161	5603	10000	74.60	62.07	343.52	0.08	2.17	250.74
100	50	50	3000	3000	428	9685	10000	1.98	6.28	42.33	0.09	1.89	31.40
100	100	100	3000	3000	330	9182	10000	11.35	30.30	173.30	0.13	3.25	126.06
100	200	200	3000	3000	157	7767	10000	51.47	125.53	685.99	0.14	4.79	501.63
200	50	50	3000	3000	399	9876	10000	4.20	13.78	86.35	0.15	3.44	63.13
200	100	100	3000	3000	238	9662	10000	29.93	58.16	343.43	0.15	5.16	252.98
200	200	200	3000	3000	157	9107	10000	135.97	225.66	1370.98	0.23	9.10	1003.73

N and vary the third one. In this part of experiments, we use $\text{Tol}_{\text{sgs}} = 10^{-5}$ to terminate our sGS-ADMM without setting the maximum iteration number. Figure 2 shows the computational results of the two algorithms over a range of m, m' or N, and each value is an average over 10 independent trials. From the results, one can see that our sGS-ADMM always returns a similar objective value as Gurobi and has a reasonably good feasibility accuracy. For the computational time, our sGS-ADMM increases approximately linearly with respect to m, m' or N individually, while Gurobi increases much more rapidly. This is because the solution methods used in Gurobi (the primal/dual simplex method and the barrier method) are no longer efficient enough and may consume too much memory (due to the Cholesky factorization of a huge coefficient matrix) when the problem size becomes large, although Gurobi already uses a parallel implementation to exploit multiple processors. Moreover, Gurobi may lack robustness, especially for solving large-scale problems. Indeed, as observed from our experiments, the computational times taken by Gurobi can vary a lot among the 10 randomly generated instances, especially m, m' or N becomes large. On the other hand, as discussed in Section 3, the main computational complexity of our sGS-ADMM at each iteration is $\mathcal{O}(Nmm')$. Hence, when two of m, m', N are fixed,



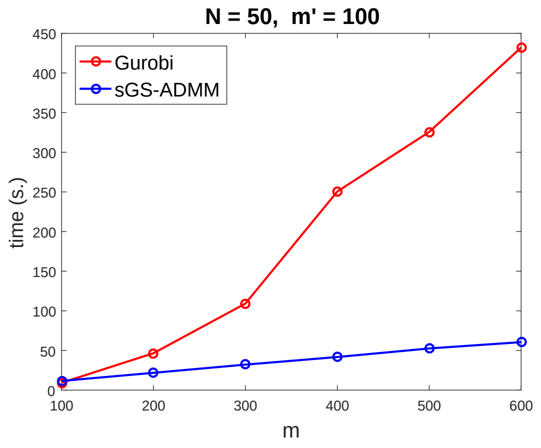
(a) Distributions and barycenters



(b) Transport plans for $n = 500$

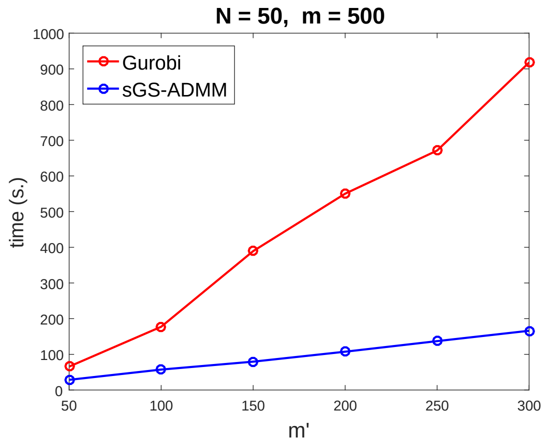
Figure 1: In figure (a): g_1 stands for the discretization of $\mathcal{N}(-2, (\frac{1}{4})^2)$; g_2 stands for the discretization of $\mathcal{N}(2, 1)$; $\bar{w}$ stands for the discretization of the true barycenter $\mathcal{N}(0, (\frac{5}{8})^2)$ of g_1 and g_2; w_1, w_2, w_3 stand for the barycenter computed by Gurobi, sGS-ADMM and BADMM, respectively; w_4, w_5, w_6 stand for the barycenter computed by IBP with $\varepsilon = 0.1, 0.01, 0.001$, respectively. The discretization is performed on the interval $[-4, 5]$ with n uniform grids. In figure (b): $\Pi^{(1)}$ (resp. $\Pi^{(2)}$) stands for the transport plan between the barycenter and g_1 (resp. g_2).

the total computational cost of our sGS-ADMM is approximately linear with respect to the remaining one, as shown in Figure 2. This then highlights another advantage of our method. In addition, although our sGS-ADMM takes advantage of many efficient built-in functions (e.g., matrix multiplication and addition) in Matlab that can execute on multiple computational threads, we believe that there is still ample room for improving our sGS-ADMM with a dedicated parallel implementation on a suitable computing platform other than Matlab. But we will leave this topic as future research.



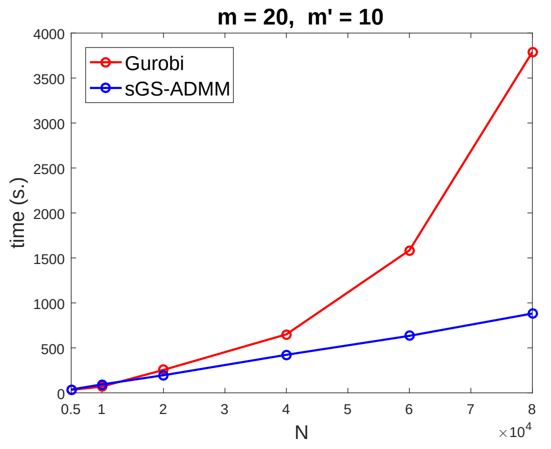
	normalized obj		feasibility	
m	Gurobi	sGS-ADMM	Gurobi	sGS-ADMM
100	0	9.19e-05	2.03e-07	1.40e-05
200	0	1.57e-04	1.25e-07	1.41e-05
300	0	3.03e-04	1.56e-07	1.40e-05
400	0	3.91e-04	1.76e-07	1.41e-05
500	0	5.20e-04	1.76e-07	1.40e-05
600	0	5.74e-04	1.71e-07	1.41e-05

(a) m varies with $N = 50$ and $m' = 100$



	normalized obj		feasibility	
m'	Gurobi	sGS-ADMM	Gurobi	sGS-ADMM
50	0	4.14e-04	2.54e-10	1.40e-05
100	0	4.85e-04	6.60e-10	1.40e-05
150	0	5.81e-04	2.56e-10	1.41e-05
200	0	6.38e-04	4.14e-10	1.42e-05
250	0	7.64e-04	1.41e-11	1.41e-05
300	0	9.64e-04	2.35e-11	1.41e-05

(b) m' varies with $N = 50$ and $m = 500$



	normalized obj		feasibility	
N	Gurobi	sGS-ADMM	Gurobi	sGS-ADMM
5000	0	1.28e-05	9.78e-09	5.19e-06
10000	0	1.12e-05	2.15e-08	4.60e-06
20000	0	1.03e-05	1.39e-08	4.19e-06
40000	0	9.83e-06	1.79e-08	3.92e-06
60000	0	9.59e-06	1.87e-08	3.77e-06
80000	0	9.35e-06	1.92e-08	3.78e-06

(c) N varies with $m = 20$ and $m' = 10$

Figure 2: Comparisons between Gurobi and sGS-ADMM

5.3 Experiments on MNIST

To better visualize the quality of results obtained by each method, we conduct similar experiments to (Cuturi and Doucet, 2014, Section 6.1) on the MNIST[4] data set (LeCun et al., 1998). Specifically, we randomly select 50 images for each digit ($0 \sim 9$) and resize each image to ζ times of its original size of 28×28, where ζ is drawn uniformly at random between 0.5 and 2. Then, we randomly put each resized image in a larger 56×56 blank image and normalize the resulting image so that all pixel values add up to 1. Thus, each image can be viewed as a discrete distribution supported on grids. We then apply sGS-ADMM, BADMM and IBP with $\varepsilon \in \{0.01, 0.001\}$ to compute a Wasserstein barycenter of the resulting images for each digit. The size of barycenter is set to 56×56. Note that, since each input image can be viewed as a *sparse* discrete distribution because most of the pixel values are zeros, one can actually solve a smaller problem (5) to obtain a barycenter; see Remark 1. Moreover, for such grid-supported data, an efficient convolutional technique (Solomon et al., 2015) and its stabilized version (Schmitz et al., 2018, Section 4.1.2) have also been used to substantially accelerate IBP and the stabilized IBP, respectively, in our experiments.

The computational results are shown in Figure 3. One can see that, our sGS-ADMM can provide a clear "smooth" barycenter just like IBP with $\varepsilon = 0.001$, although it is designed to solve the original LP. This again shows the superior quality of the solution obtained by our sGS-ADMM. Moreover, the results obtained by running sGS-ADMM for 100s are already much better than those obtained by running BADMM for 800s. IBP performs very well on this grid-supported data with smaller ε leading to sharper barycenters. Here, we would also like to point out that, without the novel convolutional technique, IBP (especially with a small ε) would take much longer time to produce sharper images. Moreover, when using the convolutional technique in IBP, one can no longer take advantage of the sparsity of the distributions and needs to solve the problem on the full grids. This may limit the adoption of the convolutional technique for the case when the distributions are highly sparse (most of weights are zeros) but supported on very dense *or* high dimensional grids. In that case, our sGS-ADMM may be more favorable.

5.4 Experiments for the Free Support Case

In this subsection, we briefly compare the performance of different methods used as subroutines in the alternating minimization method for computing a barycenter whose support points are not pre-fixed, i.e., solving problem (3); see Remark 4. The experiments are conducted on the same image data sets[5] with three different categories (mountains, sky and water) as in (Ye and Li, 2014). For each category, the data set consists of 1000 discrete distributions and each distribution is obtained by clustering pixel colors of an image in this category (Li and Wang, 2008, Section 2.3). The average number of support points is around 8 and the dimension of each support point is 3. We then compute the barycenter of each data set. The number of support points of the barycenter is chosen as $m = 10, 50$ and the initial m support points are computed as the centroids of clusters obtained by applying k-means to all the support points of the given distributions.

4. Available in `http://yann.lecun.com/exdb/mnist/`.
5. Available in `https://github.com/bobye/d2_kmeans/tree/master/data`

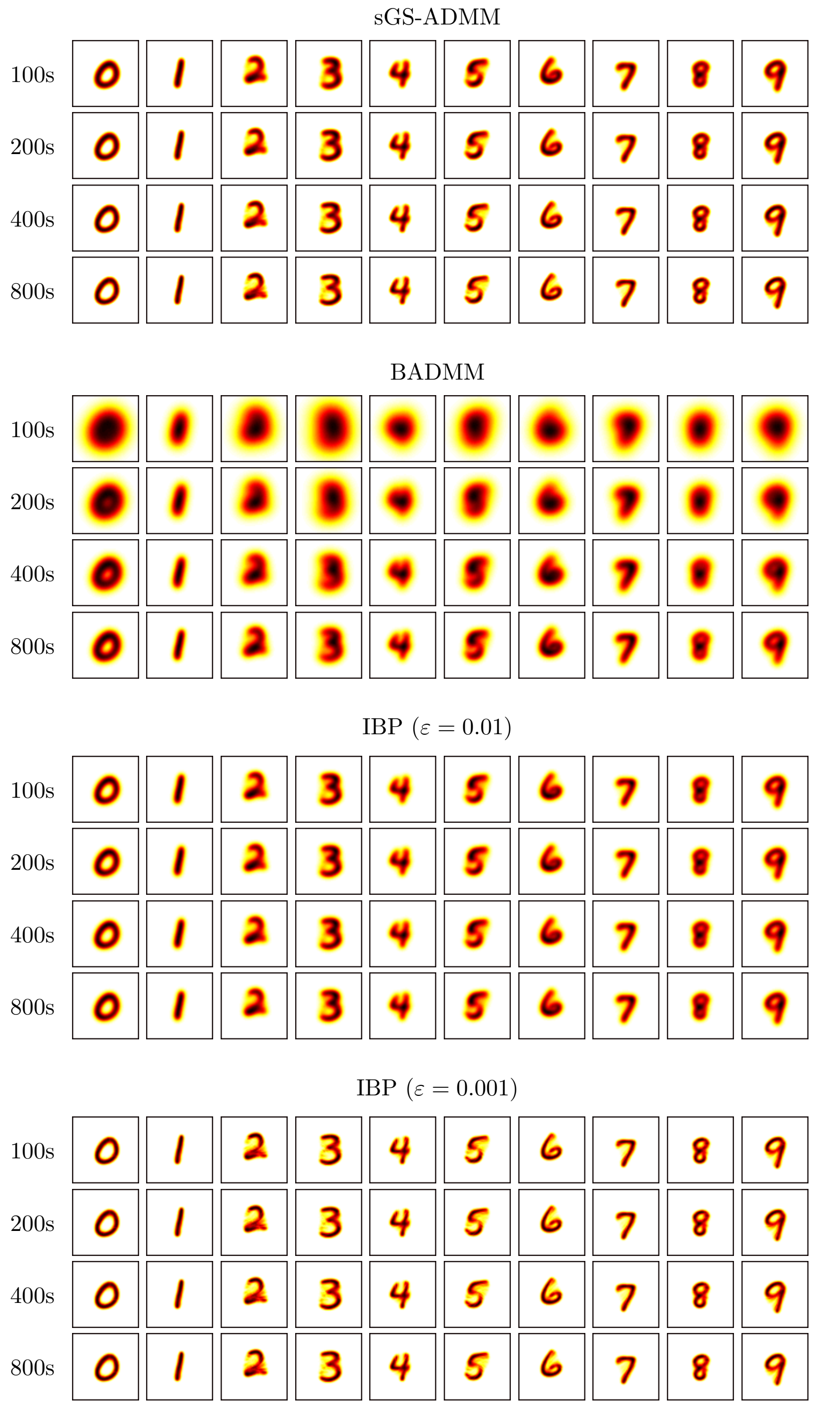


Figure 3: Barycenters obtained by running different methods for 100s, 200s, 400s, 800s, respectively.

The performance of the alternating minimization method for solving problem (3) naturally depends on the accuracy of the approximate solution obtained for each subproblem (namely, problem (4)). Basically, a more accurate approximate solution is more likely to guarantee the descent of the objective in problem (3), but is also more costly to obtain. Thus, it is nontrivial to design an optimal stopping criterion for the subroutine when solving each subproblem. In our experiments, we simply use the maximum iteration number to terminate the subroutine. For sGS-ADMM and BADMM, we follow (Ye et al., 2017) to set the maximum iteration number to 10. For IBP, we set the maximum iteration number to 100 for $\varepsilon = 0.01$ and to 1000 for $\varepsilon = 0.001$. As observed from our experiments, such maximum numbers are 'optimal' for IBP in the sense that an approximate solution having a reasonably good feasibility accuracy can be obtained in less CPU time for most cases. At each iteration, each subroutine is also warm started by the approximate solution obtained in the previous iteration. Finally, we terminate the alternating minimization and return the approximate solution when the relative successive change of the objective in problem (3) is smaller than 10^{-5}.

The computational results are reported in Table 4. One can see that Gurobi performs best in terms of the solution accuracy because it always achieves the lowest objective value and the best feasibility accuracy. However, it is significantly more time-consuming, compared with our sGS-ADMM. In comparison to BADMM and IBP, our sGS-ADMM always returns lower objective values (which are also much closer to those of Gurobi) and better feasibility accuracy within competitive computational time. Similar to the numerical observations in (Ye et al., 2017, Section IV), IBP always gives the worse objective values in our experiments. One possible reason is that, due to the entropy regularization, using IBP as the subroutine to approximately solve the subproblem is less likely to ensure the monotonic decrease of the objective values in problem (3) during the iterations. In our experiments, we have observed that careful tuning of the regularization parameter ε and intricate adjustment of the corresponding stopping criterion are needed for IBP to perform well as a subroutine within the alternating minimization method. In view of the above, our sGS-ADMM can be more favorable to be incorporated in the alternating minimization method for handling the free support case. But we should also mention that the computation of barycenters with free supports is still a challenging problem nowadays, which is nonconvex and often presented in large-scale. More in-depth study on applying our sGS-ADMM to this problem is needed and will be left as a future research project.

5.5 Summary of Experiments

From the numerical results reported in the last few subsections, one can see that our sGS-ADMM outperforms the powerful commercial solver Gurobi in terms of the computational time for solving large-scale LPs arising from Wasserstein barycenter problems. Our sGS-ADMM is also much more efficient than another non-regularization-based algorithm BADMM that is designed to solve the primal LP (6). Comparing to IBP on non-grid-supported data, our sGS-ADMM always returns high quality solutions comparable to those obtained by Gurobi but within much shorter computational time. Moreover, our sGS-ADMM is also able to find a "smooth" barycenter as in IBP even though we do not modify the LP objective function by adding an entropic regularization. Finally, we would like to

Table 4: Numerical results for the free support case

data set	method	$m = 10$			$m = 50$		
		obj	feasibility	time(s)	obj	feasibility	time(s)
mountains	Gurobi	1490.24	1.56e-12	13.46	1480.38	5.73e-11	103.39
	sGS-ADMM	1491.37	2.23e-04	2.25	1481.70	1.03e-04	6.09
	BADMM	1497.79	3.46e-03	1.21	1483.24	5.32e-03	8.73
	IBP($\varepsilon=0.01$)	1509.30	3.77e-04	3.71	1503.79	5.84e-04	10.44
	IBP($\varepsilon=0.001$)	1529.99	7.02e-04	37.20	1530.69	1.15e-03	311.93
sky	Gurobi	1623.42	1.50e-12	14.61	1612.30	1.26e-12	89.48
	sGS-ADMM	1624.60	2.98e-04	2.31	1614.27	1.12e-04	5.68
	BADMM	1632.14	3.12e-03	1.44	1633.07	2.66e-02	1.92
	IBP($\varepsilon=0.01$)	1643.13	7.62e-04	2.99	1637.36	9.06e-04	7.17
	IBP($\varepsilon=0.001$)	1735.32	2.10e-04	26.96	1639.74	1.19e-03	244.22
water	Gurobi	1620.78	3.87e-13	15.52	1611.15	5.05e-11	73.86
	sGS-ADMM	1622.20	2.02e-04	2.54	1613.16	1.31e-04	4.76
	BADMM	1653.79	8.66e-03	0.27	1615.14	1.02e-02	5.17
	IBP($\varepsilon=0.01$)	1644.79	1.34e-04	3.87	1635.35	3.77e-04	11.98
	IBP($\varepsilon=0.001$)	1734.99	1.40e-04	39.53	1646.01	2.05e-03	197.29

emphasize that in contrast to IBP that uses a small ε, our sGS-ADMM does not suffer from numerical instability issues or exceedingly slow convergence speed. Thus, one can easily apply our sGS-ADMM for computing a high quality Wasserstein barycenter without the need to implement sophisticated stabilization techniques as in the case of IBP.

6. Concluding Remarks

In this paper, we consider the problem of computing a Wasserstein barycenter with pre-specified support points for a set of discrete probability distributions with finite support points. This problem can be modeled as a large-scale linear programming (LP) problem. To solve this LP, we derive its dual problem and then adapt a symmetric Gauss-Seidel based alternating direction method of multipliers (sGS-ADMM) to solve the resulting dual problem. We also establish its global linear convergence without any condition. Moreover, we have designed the algorithm so that all the subproblems involved can be solved exactly and efficiently in a distributed fashion. This makes our sGS-ADMM highly suitable for computing a Wasserstein barycenter on a large data set. Finally, we have conducted detailed numerical experiments on synthetic data sets and image data sets to illustrate the efficiency of our method.

Acknowledgments

The authors are grateful to the editor and the anonymous referees for their valuable suggestions and comments, which have helped to improve the quality of this paper. The research of Defeng Sun was supported in part by a start-up research grant from the Hong Kong Polytechnic University. The research of Kim-Chuan Toh was supported in part by the Ministry of Education, Singapore, Academic Research Fund (Grant No. R-146-000-256-114).

Appendix A. An iterative Bregman projection method

The iterative Bregman projection (IBP) method was adapted by Benamou et al. (2015) to solve the following problem, which introduces an entropic regularization in the original LP (4):

$$\begin{aligned}\min_{\boldsymbol{w},\,\{\Pi^{(t)}\}} \quad & \tfrac{1}{N}\textstyle\sum_{t=1}^{N}\big(\langle D^{(t)},\,\Pi^{(t)}\rangle - \varepsilon E_t(\Pi^{(t)})\big)\\ \text{s.t.} \quad & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^{\top}\boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0,\quad \forall\, t = 1,\cdots,N,\\ & \boldsymbol{e}_m^{\top}\boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0,\end{aligned} \tag{19}$$

where the entropic regularization $E_t(\Pi^{(t)})$ is defined as $E_t(\Pi^{(t)}) = -\sum_{i=1}^{m}\sum_{j=1}^{m_t}\pi_{ij}^{(t)}(\log(\pi_{ij}^{(t)})-1)$ for $t = 1,\cdots,N$ and $\varepsilon > 0$ is a regularization parameter. Let $\Xi_t = \exp(-D^{(t)}/\varepsilon) \in \mathbb{R}^{m\times m_t}$ for $t = 1,\cdots,N$. Then, it follows from (Benamou et al., 2015, Remark 3) that IBP for solving (19) is given by

$$\begin{aligned}\boldsymbol{u}^{(t),k+1} &= \boldsymbol{w}^{k}./\big(\Xi_t\boldsymbol{v}^{(t),k}\big), \quad t = 1,\cdots,N,\\ \boldsymbol{v}^{(t),k+1} &= \boldsymbol{a}^{(t)}./\big(\Xi_t^{\top}\boldsymbol{u}^{(t),k+1}\big), \quad t = 1,\cdots,N,\\ \Pi^{(t),k+1} &= \mathrm{Diag}(\boldsymbol{u}^{(t),k+1})\,\Xi_t\,\mathrm{Diag}(\boldsymbol{v}^{(t),k+1}), \quad t = 1,\cdots,N,\\ \boldsymbol{w}^{k+1} &= \Big(\textstyle\prod_{t=1}^{N}\big(\boldsymbol{u}^{(t),k+1}\odot(\Xi_t\boldsymbol{v}^{(t),k+1})\big)\Big)^{\frac{1}{N}},\end{aligned} \tag{20}$$

with $\boldsymbol{w}^0 = \frac{1}{m}\boldsymbol{e}_m$ and $\boldsymbol{v}^{(t),0} = \boldsymbol{e}_{m_t}$ for $t = 1,\cdots,N$, where $\mathrm{Diag}(\boldsymbol{x})$ denotes the diagonal matrix with the vector $\boldsymbol{x}$ on the main diagonal, "./" denotes the entrywise division and "$\odot$" denotes the entrywise product. Note that the main computational cost in each iteration of the above iterative scheme is $\mathcal{O}(m\sum_{t=1}^{N}m_t)$. Moreover, when all distributions have the same m' support points, IBP can be implemented highly efficiently with a $\mathcal{O}((m+m')N)$ memory complexity, while sGS-ADMM and BADMM still require $\mathcal{O}(mm'N)$ memory. Specifically, in this case, IBP can avoid forming and storing the large matrix $[\Xi_1,\cdots,\Xi_N]$ (since each Ξ_t is the same) to compute $\Xi_t\boldsymbol{v}^{(t),k}$ and $\Xi_t^{\top}\boldsymbol{u}^{(t),k+1}$. Thus, IBP can reduce much computational cost and take less time at each iteration. This advantage can be seen in Table 3 for $\varepsilon \in \{0.1, 0.01\}$. However, we should be mindful that IBP only solves problem (19) to obtain an approximate solution of the original problem (4). Although a smaller ε can give a better approximation, IBP may become numerically unstable when ε is too small; see (Benamou et al., 2015, Section 1.3) for more details. To alleviate this numerical instability, one may carry out the computations in (20) in the log domain and use the *log-sum-exp* stabilization trick to avoid underflow/overflow for small values of ε; see (Peyré and Cuturi, 2019, Section 4.4) for more details. Specifically, by taking logarithm on both sides of the equations in (20) and letting $\tilde{\boldsymbol{u}}^{(t),k} := \varepsilon\log(\boldsymbol{u}^{(t),k})$, $\tilde{\boldsymbol{v}}^{(t),k} := \varepsilon\log(\boldsymbol{v}^{(t),k})$, $\tilde{\boldsymbol{w}}^{(t),k} := \varepsilon\log(\boldsymbol{w}^{(t),k})$ and

$\tilde{\boldsymbol{a}}^{(t)} := \varepsilon \log(\boldsymbol{a}^{(t)})$, we obtain after some manipulations that

$$\begin{aligned}
\tilde{\boldsymbol{u}}^{(t),k+1} &= \tilde{\boldsymbol{w}}^{k} + \tilde{\boldsymbol{u}}^{(t),k} - \varepsilon \log\left(\left[\sum_{j=1}^{m_t} \exp\left(\frac{\tilde{u}_i^{(t),k}+\tilde{v}_j^{(t),k}-D_{ij}^{(t)}}{\varepsilon}\right)\right]_i\right), \quad t=1,\cdots,N, \\
\tilde{\boldsymbol{v}}^{(t),k+1} &= \tilde{\boldsymbol{a}}^{(t)} + \tilde{\boldsymbol{v}}^{(t),k} - \varepsilon \log\left(\left[\sum_{i=1}^{m} \exp\left(\frac{\tilde{u}_i^{(t),k+1}+\tilde{v}_j^{(t),k}-D_{ij}^{(t)}}{\varepsilon}\right)\right]_j\right), \quad t=1,\cdots,N, \\
\Pi^{(t),k+1} &= \exp\left(\frac{\tilde{\boldsymbol{u}}^{(t),k+1}\boldsymbol{e}_{m_t}^\top+\boldsymbol{e}_m(\tilde{\boldsymbol{v}}^{(t),k+1})^\top-D^{(t)}}{\varepsilon}\right), \quad t=1,\cdots,N, \\
\tilde{\boldsymbol{w}}^{k+1} &= \frac{\varepsilon}{N}\sum_{t=1}^{N} \log\left(\left[\sum_{j=1}^{m_t} \exp\left(\frac{\tilde{u}_i^{(t),k+1}+\tilde{v}_j^{(t),k+1}-D_{ij}^{(t)}}{\varepsilon}\right)\right]_i\right),
\end{aligned} \tag{21}$$

where $\tilde{\boldsymbol{w}}^0 = \varepsilon\log(\frac{1}{m}\boldsymbol{e}_m)$ and $\tilde{\boldsymbol{u}}^{(t),0}=0$, $\tilde{\boldsymbol{v}}^{(t),0}=0$ for $t=1,\cdots,N$. After obtaining $\tilde{\boldsymbol{w}}^{k+1}$, one can recover $\boldsymbol{w}^{k+1}$ by setting $\boldsymbol{w}^{k+1} := \exp\left(\tilde{\boldsymbol{w}}^{k+1}/\varepsilon\right)$. In contrast to (20), the log-domain iterations (21) is more stable for a small ε. However, at each step, (21) requires additional exponential operations that are typically time-consuming. It also loses some computational efficiency in replacing the matrix-vector multiplications (which can take advantage of the multiprocessing capability in Matlab's Intel Math Kernel Library) in (20) by the log-sum-exp operations. Hence, iterations (21) can be much less efficient than iteration (20) in computation. This issue has also been discussed in (Peyré and Cuturi, 2019, Remark 4.23). Moreover, when ε is small, the convergence of IBP can become quite slow. In our experiments, we use (20) for $\varepsilon \in \{0.1, 0.01\}$ and use (21) for $\varepsilon = 0.001$.

Appendix B. A modified Bregman ADMM

The Bregman ADMM (BADMM) was first proposed by Wang and Banerjee (2014) and then was adapted to solve (4) by Ye et al. (2017). For notational simplicity, let

$$\begin{aligned}
\mathcal{C}_1 &:= \{(\Pi^{(1)},\cdots,\Pi^{(N)}) \,:\, (\Pi^{(t)})^\top\boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)}\geq 0,\ t=1,\cdots,N\}, \\
\mathcal{C}_2 &:= \{(\Gamma^{(1)},\cdots,\Gamma^{(N)},\boldsymbol{w}) \,:\, \boldsymbol{w}\in\Delta_m,\ \Gamma^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ \Gamma^{(t)}\geq 0,\ t=1,\cdots,N\}.
\end{aligned}$$

Then, problem (4) can be equivalently rewritten as

$$\begin{aligned}
\min_{\{\Pi^{(t)}\},\,\{\Gamma^{(t)}\},\,\boldsymbol{w}} \quad & \textstyle\sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\
\text{s.t.} \quad & \Pi^{(t)} = \Gamma^{(t)}, \quad t=1,\cdots,N, \\
& (\Pi^{(1)},\cdots,\Pi^{(N)})\in\mathcal{C}_1, \quad (\Gamma^{(1)},\cdots,\Gamma^{(N)},\boldsymbol{w})\in\mathcal{C}_2.
\end{aligned} \tag{22}$$

The iterative scheme of BADMM for solving (22) is given by

$$\begin{cases}
(\Pi^{(1),k+1},\cdots,\Pi^{(N),k+1}) = \underset{(\Pi^{(1)},\cdots,\Pi^{(N)})\in\mathcal{C}_1}{\operatorname{argmin}} \left\{\sum_{t=1}^N\left(\langle D^{(t)},\,\Pi^{(t)}\rangle + \langle\Lambda^{(t),k},\,\Pi^{(t)}\rangle + \rho\mathbf{KL}(\Pi^{(t)},\,\Gamma^{(t),k})\right)\right\}, \\
(\Gamma^{(1),k+1},\cdots,\Gamma^{(N),k+1},\boldsymbol{w}^{k+1}) = \underset{(\Gamma^{(1)},\cdots,\Gamma^{(N)},\boldsymbol{w})\in\mathcal{C}_2}{\operatorname{argmin}} \left\{\sum_{t=1}^N\left(-\langle\Lambda^{(t),k},\,\Gamma^{(t)}\rangle + \rho\mathbf{KL}(\Gamma^{(t)},\,\Pi^{(t),k+1})\right)\right\}, \\
\Lambda^{(t),k+1} = \Lambda^{(t),k} + \rho(\Pi^{(t),k+1} - \Gamma^{(t),k+1}), \quad t=1,\cdots,N,
\end{cases}$$

where $\mathbf{KL}(\cdot,\cdot)$ denotes the KL divergence defined by $\mathbf{KL}(A,B)=\sum_{ij}a_{ij}\ln(\frac{a_{ij}}{b_{ij}})$ for any two matrices A, B of the same size. The subproblems in above scheme have closed-form solutions; see (Ye et al., 2017, Section III.B) for more details. Indeed, at the k-th iteration,

$$
\begin{aligned}
\boldsymbol{u}^{(t),k} &= \left(\frac{a_j^{(t)}}{(\Gamma_{:j}^{(t),k})^\top \exp(-\frac{1}{\rho}D_{:j}^{(t)}-\frac{1}{\rho}\Lambda_{:j}^{(t),k})}\right)_{j=1,\cdots,m_t}, \quad t=1,\cdots,N,\\
\Pi^{(t),k+1} &= \left(\Gamma^{(t),k}\odot\exp(-\tfrac{1}{\rho}D^{(t)}-\tfrac{1}{\rho}\Lambda^{(t),k})\right)\mathrm{Diag}(\boldsymbol{u}^{(t),k}), \quad t=1,\cdots,N,\\
\tilde{\boldsymbol{w}}^{(t),k+1} &= \left((\Pi_{i:}^{(t),k+1})^\top \exp(\tfrac{1}{\rho}\Lambda_{i:}^{(t),k})\right)_{i=1,\cdots,m}, \quad t=1,\cdots,N,\\
\boldsymbol{w}^{k+1} &= \left(\textstyle\prod_{t=1}^N\tilde{\boldsymbol{w}}^{(t),k+1}\right)^{\frac{1}{N}} \Big/ \left(\boldsymbol{e}_m^\top\left(\textstyle\prod_{t=1}^N\tilde{\boldsymbol{w}}^{(t),k+1}\right)^{\frac{1}{N}}\right),\\
\boldsymbol{v}^{(t),k+1} &= \left(\frac{w_i^{k+1}}{(\Pi_{i:}^{(t),k+1})^\top\exp(\frac{1}{\rho}\Lambda_{i:}^{(t),k})}\right)_{i=1,\cdots,m}, \quad t=1,\cdots,N,\\
\Gamma^{(t),k+1} &= \mathrm{Diag}(\boldsymbol{v}^{(t),k+1})\left(\Pi^{(t),k+1}\odot\exp(\tfrac{1}{\rho}\Lambda^{(t),k})\right), \quad t=1,\cdots,N.
\end{aligned}
$$

Moreover, in order to avoid computing the geometric mean $(\prod_{t=1}^N\tilde{\boldsymbol{w}}^{(t),k+1})^{\frac{1}{N}}$ for updating $\boldsymbol{w}^{k+1}$, Ye et al. (2017) actually use one of the following heuristic rules to update $\boldsymbol{w}^{k+1}$:

$$
\begin{aligned}
&\text{(R1)} \qquad \boldsymbol{w}^{k+1} = \left(\textstyle\sum_{t=1}^N\tilde{\boldsymbol{w}}^{(t),k+1}\right)\Big/\left(\boldsymbol{e}_m^\top\left(\textstyle\sum_{t=1}^N\tilde{\boldsymbol{w}}^{(t),k+1}\right)\right),\\
&\text{(R2)} \qquad \boldsymbol{w}^{k+1} = \left(\textstyle\sum_{t=1}^N\sqrt{\tilde{\boldsymbol{w}}^{(t),k+1}}\right)^2\Big/\left(\boldsymbol{e}_m^\top\left(\textstyle\sum_{t=1}^N\sqrt{\tilde{\boldsymbol{w}}^{(t),k+1}}\right)^2\right).
\end{aligned}
$$

In their Matlab codes, (R2) is the default updating rule. The main computational complexity without considering the exponential operations in BADMM is $\mathcal{O}(m\sum_{t=1}^N m_t)$. For the exponential operations at each step, the practical computational cost could be a few times more than the previous cost of $\mathcal{O}(m\sum_{t=1}^N m_t)$.

References

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

E. Anderes, S. Borgwardt, and J. Miller. Discrete Wasserstein barycenters: Optimal transport for discrete data. *Mathematical Methods of Operations Research*, 84(2):389–409, 2016.

H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, volume 408. Springer, 2011.

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.

J. Bigot and T. Klein. Characterization of barycenters in the Wasserstein space by averaging optimal transport maps. *ESAIM: Probability and Statistics*, 22:35–57, 2018.

S. Borgwardt. An LP-based, strongly polynomial 2-approximation algorithm for sparse Wasserstein barycenters. *To appear in Operational Research*, 2020.

S. Borgwardt and S. Patterson. Improved linear programs for discrete barycenters. *INFORMS Journal on Optimization*, 2(1):14–33, 2020.

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.

C. Chen, B. He, Y. Ye, and X. Yuan. The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1): 57–79, 2016.

L. Chen, D. F. Sun, and K.-C. Toh. An efficient inexact symmetric Gauss-Seidel based majorized ADMM for high-dimensional convex composite conic programming. *Mathematical Programming*, 161(1-2):237–270, 2017.

L. Chen, X. Li, D. F. Sun, and K.-C. Toh. On the equivalence of inexact proximal ALM and ADMM for a class of convex composite programming. *To appear in Mathematical Programming*, 2019.

S. Claici, E. Chien, and J. Solomon. Stochastic Wasserstein barycenters. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 999–1008, 2018.

L. Condat. Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1-2):575–585, 2016.

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

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

M. Cuturi and G. Peyré. A smoothed dual approach for variational Wasserstein problems. *SIAM Journal on Imaging Sciences*, 9(1):320–343, 2016.

J. A. De Loera and E. D. Kim. Combinatorics and geometry of transportation polytopes: An update. *Contemporary Mathematics*, 625:37–76, 2014.

A. Dessein, N. Papadakis, and J.-L. Rouas. Regularized optimal transport and the rot mover's distance. *Journal of Machine Learning Research*, 19(1):590–642, 2018.

M. Essid and J. Solomon. Quadratically regularized optimal transport on graphs. *SIAM Journal on Scientific Computing*, 40(4):A1961–A1986, 2018.

M. Fazel, T. K. Pong, D. F. Sun, and P. Tseng. Hankel matrix rank minimization with applications to system identification and realization. *SIAM Journal on Matrix Analysis and Applications*, 34(3):946–977, 2013.

D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Computers & Mathematics with Applications*, 2(1):17–40, 1976.

R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Revue Francaise d'Automatique, Informatique, Recherche Opérationelle*, 9(R-2):41–76, 1975.

Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2018. URL `http://www.gurobi.com`.

D. Han, D. F. Sun, and L. Zhang. Linear rate convergence of the alternating direction method of multipliers for convex composite programming. *Mathematics of Operations Research*, 43(2):622–637, 2018.

B. He, M. Tao, and X. Yuan. Alternating direction method with Gaussian back substitution for separable convex programming. *SIAM Journal on Optimization*, 22(2):313–340, 2012.

X. Y. Lam, J. S. Marron, D. F. Sun, and K.-C. Toh. Fast algorithms for large scale generalized distance weighted discrimination. *Journal of Computational and Graphical Statistics*, 27(2):368–379, 2018.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Li and J. Z. Wang. Real-time computerized annotation of pictures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):985–1002, 2008.

M. Li, D. F. Sun, and K.-C. Toh. A convergent 3-block semi-proximal ADMM for convex minimization problems with one strongly convex block. *Asia-Pacific Journal of Operational Research*, 32(3):1550024(19p), 2015.

X. Li, D. F. Sun, and K.-C. Toh. A Schur complement based semi-proximal ADMM for convex quadratic conic programming and extensions. *Mathematical Programming*, 155 (1-2):333–373, 2016.

X. Li, D. F. Sun, and K.-C. Toh. QSDPNAL: A two-phase augmented lagrangian method for convex quadratic semidefinite programming. *Mathematical Programming Computation*, 10(4):703–743, 2018.

R.J. McCann. A convexity principle for interacting gases. *Advances in Mathematics*, 128 (1):153–179, 1997.

A.M. Oberman and Y. Ruan. An efficient linear programming method for optimal transportation. *arXiv preprint arXiv: 1509.03668*, 2015.

G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

J. Rabin, G. Peyré, J. Delon, and M. Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 435–446, 2011.

S. M. Robinson. Some continuity properties of polyhedral multifunctions. *Mathematical Programming at Oberwolfach, vol.14 of Mathematical Programming Studies*, pages 206–214, 1981.

R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, 1970.

R. T. Rockafellar and R. J-B. Wets. *Variational Analysis.* Springer, 1998.

A. Ruszczyński. *Nonlinear Optimization.* Princeton University Press, Princeton, 2006.

M.A. Schmitz, M. Heitz, N. Bonneel, F. Ngole, D. Coeurjolly, M. Cuturi, G. Peyré, and J.-L. Starck. Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning. *SIAM Journal on Imaging Sciences*, 11(1):643–678, 2018.

B. Schmitzer. Stabilized sparse scaling algorithms for entropy regularized transport problems. *SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

J. Solomon, F. De Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics*, 34(4):1–11, 2015.

D. F. Sun, K.-C. Toh, and L. Yang. A convergent 3-block semiproximal alternating direction method of multipliers for conic programming with 4-type constraints. *SIAM Journal on Optimization*, 25(2):882–915, 2015.

C. A. Uribe, D. Dvinskikh, P. Dvurechensky, A. Gasnikov, and A. Nedić. Distributed computation of Wasserstein barycenters over networks. In *IEEE Conference on Decision and Control*, pages 6544–6549, 2018.

Cédric Villani. *Optimal Transport: Old and New*, volume 338. Springer Science & Business Media, 2008.

H. Wang and A. Banerjee. Bregman alternating direction method of multipliers. In *Advances in Neural Information Processing Systems*, pages 2816–2824, 2014.

Y. Xie, X. Wang, R. Wang, and H. Zha. A fast proximal point method for computing exact Wasserstein distance. In *Proceedings of Machine Learning Research*, volume 115, pages 433–453, 2020.

J. Ye and J. Li. Scaling up discrete distribution clustering using ADMM. In *IEEE International Conference on Image Processing*, pages 5267–5271, 2014.

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.