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

Lei Yang* Jia Li† Defeng Sun‡ Kim-Chuan Toh§

September 13, 2018

Abstract

In this paper, we consider the problem of computing a Wasserstein barycenter for a set of discrete probability distributions with finite supports, which finds many applications in different 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), while the problem size can be extremely large. To handle this large-scale LP, in this paper, we derive 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 analyze its global convergence as well as its global linear convergence rate. 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 dataset. 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 datasets and image datasets demonstrate that our method is more efficient for solving large-scale problems, comparing with 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 [1]. Recently, the Wasserstein barycenter has attracted much attention due to its promising performance in many application areas such as data analysis and statistics [5], machine learning [13, 23, 36, 37] and image processing [27]. 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 [2]. 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, the classical LP

*Institute of Operations Research and Analytics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076. (orayl@nus.edu.sg).

†Department of Statistics, Pennsylvania State University, University Park, PA 16802, USA. (jiali@stat.psu.edu).

‡Department of Applied Mathematics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. (defeng.sun@polyu.edu.hk). The research of this author was supported in part by a start-up research grant from the Hong Kong Polytechnic University.

§Department of Mathematics, and Institute of Operations Research and Analytics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076. (mattohkc@nus.edu.sg). The research of this author was supported in part by the Ministry of Education, Singapore, Academic Research Fund (Grant No. R-146-000-256-114).

methods such as the simplex method and the interior point method are no longer efficient 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, [4, 6, 7, 11, 13, 33, 36, 37].

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 [13] and the iterative Bregman projection (IBP) method [4], to solve the regularized problem. These methods can be implemented efficiently and hence are suitable for large-scale datasets. However, they can only return an approximate solution of the LP and often encounter numerical issues when the regularization parameter becomes small. 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 [36]. However, solving the quadratic programming subproblems involved is still highly expensive. Later, Ye at al. [37] developed a modified Bregman ADMM (BADMM) based on the original one [35] to solve the LP. In this method, all subproblems have closed-form solutions and hence can be solved efficiently. Promising numerical performance is also reported in [37]. 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. Our method is actually 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 [10, 25]. 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 [16, 17]. The 2-block ADMM can be naturally extended to a multi-block ADMM for solving a convex problem with more than 2 blocks of variables. However, it has been shown in [8] that the directly extended ADMM may not converge when directly applied to a convex problem with 3 or more blocks of variables. This inspires many researchers to develop various convergent variants of the ADMM for convex problems with more than 2 blocks of variables; see, for example, [9, 10, 20, 25, 24, 32]. Among them, the Schur complement based convergent semi-proximal ADMM (sPADMM) was proposed by Li et al. [25] 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 essentially follows the original ADMM, but performs 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. [25] show that their method can be reformulated as a 2-block sPADMM with specially designed semi-proximal terms and its convergence is guaranteed from that of the 2-block sPADMM; see [15, Appendix B]. Later, this method was generalized to the inexact symmetric Gauss-Seidel based ADMM (sGS-ADMM) for more general convex problems [10, 26]. The numerical results reported in [10, 25, 26] also show 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 [19, 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 as follows:

1. We derive the dual problem of the original primal LP and characterize the properties of their optimal solutions; see Proposition 4.1. The resulting dual problem is our target problem, which is a linearly constrained convex problem containing 3 blocks of variables with a nice separable structure. We should emphasize again that we do not introduce any entropy regularization to modify the LP so as to make it computationally more tractable. This is in contrast to most existing works (e.g., [4, 6, 13, 33, 36, 37]) that primarily focus on (approximately) solving the original primal LP.

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 4.1 and 4.2. We also

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 dataset.

3. We conduct rigorous numerical experiments on synthetic datasets and MNIST 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 performs much better than IBP and BADMM, and is also able to outperform Gurobi in 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. A simple extension to the free support case is discussed in Section 5. Finally, numerical results are presented in Section 6, with some concluding remarks given in Section 7.

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 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 $\mathrm{vex}(X)$ 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 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. For a proper closed convex function $f : \mathbb{R}^n \to (-\infty, \infty]$, its subdifferential at $\boldsymbol{x} \in \mathrm{dom}\, f$ is $\partial f(\boldsymbol{x}) := \{\boldsymbol{d} \in \mathbb{R}^n : f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{d}, \boldsymbol{y} - \boldsymbol{x}\rangle,\ \forall\, \boldsymbol{y} \in \mathbb{R}^n\}$ 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 [29, Theorem 23.5] that

$$\boldsymbol{y} \in \partial f(\boldsymbol{x}) \iff \boldsymbol{x} \in \partial f^*(\boldsymbol{y}). \tag{1.1}$$

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.

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 $a_i \geq 0$, $i = 1, \cdots, m$.

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 [34, Chapter 6] for more details on the Wasserstein distance and to [1, 2] 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 2-Wasserstein distance between $\mathcal{P}^{(u)}$ and $\mathcal{P}^{(v)}$ is defined by $\mathcal{W}_2(\mathcal{P}^{(u)}, \mathcal{P}^{(v)}) := \sqrt{\mathrm{v}^*}$,

where v^* is the optimal objective value of the following linear programming:

$$\mathrm{v}^* := \min_{\pi_{ij}\geq 0} \left\{ \sum_i^{m_u}\sum_j^{m_v} \pi_{ij}\|\boldsymbol{q}_i^{(u)} - \boldsymbol{q}_j^{(v)}\|^2 \ : \ \begin{array}{l} \sum_{i=1}^{m_u}\pi_{ij} = a_j^{(v)}, \ \ \forall\, j = 1,\cdots,m_v \\ \sum_{j=1}^{m_v}\pi_{ij} = a_i^{(u)}, \ \ \forall\, i = 1,\cdots,m_u \end{array} \right\}.$$

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 Wasserstein barycenter $\mathcal{P} := \{(w_i,\, \boldsymbol{x}_i) : i = 1,\cdots,m\}$ with m support points (m is pre-specified empirically) is an optimal solution of the following problem

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

This is a two-stage optimization problem that can be easily shown to be equivalent to

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\, X,\, \{\Pi^{(t)}\}} & \sum_{t=1}^N \langle \mathcal{D}(X,\, Q^{(t)}),\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0, \ \ \forall\, t = 1,\cdots,N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{2.1}$$

where

- $\boldsymbol{e}_{m_t}$ (resp. $\boldsymbol{e}_m$) denotes the m_t (resp. m) dimensional vector with all entries being 1;
- $\boldsymbol{w} := (w_1,\cdots,w_m)^\top \in \mathbb{R}_+^m$, $X := [\boldsymbol{x}_1,\cdots,\boldsymbol{x}_m] \in \mathbb{R}^{d\times m}$;
- $\boldsymbol{a}^{(t)} := (a_1^{(t)},\cdots,a_{m_t}^{(t)})^\top \in \mathbb{R}_+^{m_t}$, $Q^{(t)} := [\boldsymbol{q}_1^{(t)},\cdots,\boldsymbol{q}_{m_t}^{(t)}] \in \mathbb{R}^{d\times m_t}$ for $t = 1,\cdots,N$;
- $\Pi^{(t)} = [\,\pi_{ij}^{(t)}\,] \in \mathbb{R}^{m\times m_t}$, $\mathcal{D}(X,\, Q^{(t)}) := [\,\|\boldsymbol{x}_i - \boldsymbol{q}_j^{(t)}\|^2\,] \in \mathbb{R}^{m\times m_t}$ for $t = 1,\cdots,N$.

Note that (2.1) 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 $\{\mathcal{P}^{(t)}\}_{t=1}^N$. Indeed, in some cases, all distributions in $\{\mathcal{P}^{(t)}\}_{t=1}^N$ have the same set of support points and hence the barycenter should also take the same set of support points. 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 (2.1) reduces to the following problem:

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\, \{\Pi^{(t)}\}} & \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0, \ \ \forall\, t = 1,\cdots,N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{2.2}$$

where $D^{(t)}$ denotes $\mathcal{D}(X,\, Q^{(t)})$ for simplicity. This is also the main problem studied in [4, 6, 7, 11, 13, 33, 36, 37] for the computation of Wasserstein barycenters. Moreover, one can easily see that (2.2) is indeed a large-scale LP containing $(m+m\sum_{t=1}^N m_t)$ variables with nonnegative constraints and $(Nm+\sum_{t=1}^N m_t+1)$ equality constraints. For $N = 100$, $m = 1000$ and $m_t = 1000$ for all $t = 1,\ldots,N$, the LP has about 10^8 variables and 2×10^5 equality constraints.

Remark 2.1 (**Practical computational consideration when $\boldsymbol{a}^{(t)}$ is sparse**). *Note that any feasible point $(\boldsymbol{w},\, \{\Pi^{(t)}\})$ of (2.2) 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 $a_j^{(t)} = 0$ for some $1 \leq j \leq m_t$ and $1 \leq t \leq N$, then $\pi_{ij}^{(t)} = 0$ 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* (2.2)*, the point* $(\boldsymbol{w}^*, \{\Pi^{(t),*}_{\mathcal{J}_t}\})$ *is also an optimal solution of the following problem*

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\,\{\hat{\Pi}^{(t)}\}} & \sum_{t=1}^N \langle D^{(t)},\, \hat{\Pi}^{(t)}\rangle \\ \text{s.t.} & \hat{\Pi}^{(t)}\boldsymbol{e}_{m'_t} = \boldsymbol{w},\ (\hat{\Pi}^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)}_{\mathcal{J}_t},\ \hat{\Pi}^{(t)} \geq 0,\quad \forall\, t = 1, \cdots, N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{2.3}$$

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 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* $(\boldsymbol{w}^*, \{\hat{\Pi}^{(t),*}\})$ *of* (2.3)*, the point* $(\boldsymbol{w}^*, \{\Pi^{(t),*}\})$ *obtained by setting* $\Pi^{(t),*}_{\mathcal{J}_t} = \hat{\Pi}^{(t),*}$ *and* $\Pi^{(t),*}_{\mathcal{J}^c_t} = 0$ *is also an optimal solution of* (2.2)*, where* $\mathcal{J}^c_t := \{\, j : a^{(t)}_j = 0 \,\}$*.*

Therefore, one can obtain an optimal solution of (2.2) *by computing an optimal solution of* (2.3)*. Note that the problem size of* (2.3) *can be much smaller than that of* (2.2) *when each* $\boldsymbol{a}^{(t)}$ *is sparse, i.e.,* $m'_t \ll m_t$*. Thus, solving* (2.3) *can reduce the computational cost and save memory in practice. Since* (2.3) *takes the same form as* (2.2)*, we only consider* (2.2) *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 $t = 1, \cdots, N$. Then, (2.2) can be equivalently written as

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\,\{\Pi^{(t)}\}} & \delta_{\Delta_m}(\boldsymbol{w}) + \sum_{t=1}^N \delta^t_+(\Pi^{(t)}) + \sum_{t=1}^N \langle D^{(t)},\, \Pi^{(t)}\rangle \\ \text{s.t.} & \Pi^{(t)}\boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\quad \forall\, t = 1, \cdots, N. \end{array} \tag{2.4}$$

We next derive the dual problem of (2.4) (hence (2.2)). To this end, we write down the Lagrangian function associated with (2.4) as follows:

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

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 (2.4) is given by

$$\max_{\{\boldsymbol{y}^{(t)}\},\,\{\boldsymbol{z}^{(t)}\}} \left\{ \min_{\boldsymbol{w},\,\{\Pi^{(t)}\}} \left\{ \Upsilon\left(\boldsymbol{w}, \{\Pi^{(t)}\}; \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\right) \right\} \right\}. \tag{2.5}$$

Observe that

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

where $\delta^*_{\Delta_m}$ is the Fenchel conjugate of δ_{Δ_m}. Thus, (2.5) is equivalent to

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

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

$$\begin{array}{ll} \min\limits_{\boldsymbol{u},\,\{V^{(t)}\},\,\{\boldsymbol{y}^{(t)}\},\,\{\boldsymbol{z}^{(t)}\}} & \delta^*_{\Delta_m}(\boldsymbol{u}) + \sum_{t=1}^N \delta_+^t(V^{(t)}) + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle \\ \text{s.t.} & \sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u} = 0, \\ & V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top = 0, \quad t = 1, \cdots, N. \end{array} \tag{2.6}$$

Note that (2.6) can be viewed as a linearly constrained convex problem with 3 blocks of variables grouped as $(\boldsymbol{u}, V^{(1)}, \cdots, V^{(N)})$, $(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(N)})$ and $(\boldsymbol{z}^{(1)}, \cdots, \boldsymbol{z}^{(N)})$, whose objective is nonsmooth only with respect to $(\boldsymbol{u}, V^{(1)}, \cdots, V^{(N)})$ 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 [10, 25]. Then, it is natural to adapt the sGS-ADMM for solving (2.6), which is presented in the next section.

Remark 2.2 (**2-block ADMM for solving** (2.2))**.** *It is worth noting that one can also apply the 2-block ADMM to solve the primal problem* (2.2) *by introducing some proper auxiliary variables. For example, one can consider the following equivalent reformulation of* (2.2)*:*

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

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* $(\Gamma^{(1)}, \cdots, \Gamma^{(N)})$ *being the other one. This 2-block ADMM avoids solving the quadratic programming subproblems and hence is more efficient than the one used in [36]. 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 *becomes 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* (2.2)*, but they all perform worse than our sGS-ADMM presented later. Hence, we will no longer consider ADMM-type methods for solving the primal problem* (2.2) *or its equivalent variants in this paper.*

3 sGS-ADMM for computing Wasserstein barycenters

In this section, we present the sGS-ADMM for solving (2.6). First, we write down the augmented Lagrangian function associated with (2.6) as follows:

$$\begin{aligned} &\mathcal{L}_\beta\Big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}, \{\Lambda^{(t)}\}\Big) \\ &= \delta^*_{\Delta_m}(\boldsymbol{u}) + \sum_{t=1}^N \delta_+^t(V^{(t)}) + \sum_{t=1}^N \langle \boldsymbol{z}^{(t)},\, \boldsymbol{a}^{(t)}\rangle + \langle \boldsymbol{\lambda},\, \sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}\rangle + \sum_{t=1}^N \langle \Lambda^{(t)},\, V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\rangle \\ &\quad + \frac{\beta}{2}\|\sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}\|^2 + \frac{\beta}{2}\sum_{t=1}^N \|V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top\|_F^2, \end{aligned}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$, $\Lambda^{(t)} \in \mathbb{R}^{m\times m_t}$, $t = 1, \cdots, N$ are multipliers and $\beta > 0$ is the penalty parameter. The sGS-ADMM for solving (2.6) is presented 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. In the next section, we shall see that $\big(\{\boldsymbol{y}^{(t),k+1}\}, \{\boldsymbol{z}^{(t),k+1}\}\big)$ computed from **Step 2a**–**2c** can be exactly obtained by minimizing $\mathcal{L}_\beta$ plus a special proximal term with respect to $\big(\{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}\big)$. Moreover, 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.

Algorithm 1 sGS-ADMM for solving (2.6)

Input: the penalty parameter $\beta > 0$, the dual step-size $\tau \in (0, \frac{1+\sqrt{5}}{2})$ 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 $\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}\big) = \arg\min\limits_{\boldsymbol{u},\, \{V^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}, \{V^{(t)}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.

Step 2a. Compute $\{\tilde{\boldsymbol{z}}^{(t),k+1}\} = \arg\min\limits_{\{\boldsymbol{z}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.

Step 2b. Compute $\{\boldsymbol{y}^{(t),k+1}\} = \arg\min\limits_{\{\boldsymbol{y}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t)}\}, \{\tilde{\boldsymbol{z}}^{(t),k+1}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.

Step 2c. Compute $\{\boldsymbol{z}^{(t),k+1}\} = \arg\min\limits_{\{\boldsymbol{z}^{(t)}\}} \mathcal{L}_\beta\big(\boldsymbol{u}^{k+1}, \{V^{(t),k+1}\}, \{\boldsymbol{y}^{(t),k+1}\}, \{\boldsymbol{z}^{(t)}\}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)$.

Step 3. Compute

$$\begin{aligned}\boldsymbol{\lambda}^{k+1} &= \boldsymbol{\lambda}^k + \tau\beta\big(\textstyle\sum_{t=1}^N \boldsymbol{y}^{(t),k+1} - \boldsymbol{u}^{k+1}\big),\\ \Lambda^{(t),k+1} &= \Lambda^{(t),k} + \tau\beta\big(V^{(t),k+1} - D^{(t)} - \boldsymbol{y}^{(t),k+1}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t),k+1})^\top\big), \quad t = 1, \cdots, N.\end{aligned}$$

end while

Output: $\boldsymbol{u}^{k+1}$, $\{V^{(t),k+1}\}$, $\{\boldsymbol{y}^{(t),k+1}\}$, $\{\boldsymbol{z}^{(t),k+1}\}$, $\boldsymbol{\lambda}^{k+1}$, $\{\Lambda^{(t),k+1}\}$.

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 shall see shortly 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**.

Before ending this section, we present the computational details and the efficient implementations in each step of Algorithm 1.

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

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

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 (see [3, 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 [12] with the complexity of $\mathcal{O}(m)$ that is typically observed in practice. Moreover, for each $t = 1, \cdots, N$, $V^{(t),k+1}$ is obtained 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 very large, even performing simple operations such as adding two such matrices can be time consuming. Thus we have paid special attention to design 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 $\tilde{\boldsymbol{z}}^{(1),k+1}$, $\cdots$, $\tilde{\boldsymbol{z}}^{(N),k+1}$ independently. 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 $t = 1, \ldots, N$. The computational complexity in this step is $\mathcal{O}(m\sum_{t=1}^N m_t)$.

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 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 conditions, namely, $\nabla\Phi^k(\boldsymbol{y}^{(1),k+1}, \cdots, \boldsymbol{y}^{(N),k+1}) = 0$ that

$$\textstyle\sum_{\ell=1}^N (\boldsymbol{y}^{(\ell),k+1} - \boldsymbol{y}^{(\ell),k}) + m_t(\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k}) + \boldsymbol{h}^k + \widetilde{B}^{(t),k}\boldsymbol{e}_{m_t} = 0, \quad \forall\, t = 1, \cdots, N. \tag{3.1}$$

By dividing m_t in (3.1) 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 and (3.1), 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 $t = 1, \ldots, N$. 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 $t = 1, \cdots, N$. Observe that 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 $t = 1, \cdots, N$, $\boldsymbol{z}^{(t),k+1}$ is obtained independently by solving

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

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 $C^{(t),k} := D^{(t)} + \boldsymbol{y}^{(t),k+1}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t),k})^\top - \beta^{-1}\Lambda^{(t),k} - V^{(t),k+1} = B^{(t),k} + (\boldsymbol{y}^{(t),k+1} - \boldsymbol{y}^{(t),k})\boldsymbol{e}_{m_t}^\top$. Based on the above, one can also compute $\boldsymbol{z}^{(t),k+1}$ very efficiently. The computational complexity in this step is $\mathcal{O}(Nm + \sum_{t=1}^N m_t)$, which is much smaller than the cost in **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 $\mathcal{O}(m\sum_{t=1}^N m_t)$.

4 Convergence analysis

In this section, we shall establish the global linear convergence of Algorithm 1 based on the convergence results developed in [15, 19, 25]. To this end, we first write down the KKT system associated with (2.4) as follows:

$$\begin{aligned}
0 &\in \partial\delta_{\Delta_m}(\boldsymbol{w}) - \big(\textstyle\sum_{t=1}^N \boldsymbol{y}^{(t)}\big), \\
0 &\in \partial\delta_+^t(\Pi^{(t)}) + D^{(t)} + \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top + \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top, \quad \forall\, t = 1, \cdots, N, \\
0 &= \Pi^{(t)}\boldsymbol{e}_{m_t} - \boldsymbol{w}, \quad \forall\, t = 1, \cdots, N, \\
0 &= (\Pi^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)}, \quad \forall\, t = 1, \cdots, N.
\end{aligned} \tag{4.1}$$

We also write down the KKT system associated with (2.6) as follows:

$$\begin{aligned}
0 &\in \partial\delta^*_{\Delta_m}(\boldsymbol{u}) - \boldsymbol{\lambda}, \\
0 &\in \partial\delta_+^t(V^{(t)}) + \Lambda^{(t)}, \quad \forall\, t = 1, \cdots, N, \\
0 &= \Lambda^{(t)}\boldsymbol{e}_{m_t} - \boldsymbol{\lambda}, \quad \forall\, t = 1, \cdots, N, \\
0 &= (\Lambda^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)}, \quad \forall\, t = 1, \cdots, N, \\
0 &= \textstyle\sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}, \\
0 &= V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)}\boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m(\boldsymbol{z}^{(t)})^\top, \quad \forall\, t = 1, \cdots, N.
\end{aligned} \tag{4.2}$$

Then, we show the existence of optimal solutions of problems (2.4) and (2.6), and their relations in the following proposition.

Proposition 4.1. *The following statements hold.*

(i) *The optimal solution of* (2.4) *exists and the solution set of the KKT system* (4.1) *is nonempty;*

(ii) *The optimal solution of* (2.6) *exists and the solution set of the KKT system* (4.2) *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* (4.2)*, then* $(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ *solves* (2.6) *and* $\big(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ *solves* (2.4).

Proof. *Statement* (i). Note that (2.4) is equivalent to (2.2). Thus, we only need to show that the optimal solution of (2.2) exists. To this end, we first claim that the feasible set of (2.2) is nonempty. For simplicity, let

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

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 [14, Lemma 2.2] that each $\Omega^t(\bar{\boldsymbol{w}})$ is nonempty. Hence, $\mathcal{C}_{\rm feas}$ is nonempty. Moreover, it is not hard to see that $\mathcal{C}_{\rm feas}$ is closed and bounded. This together with the continuity of the objective function in (2.2) implies that the optimal solution of (2.2) exists. Hence, the optimal solution of (2.4) exists. Now, let $(\boldsymbol{w}^*, \{\Pi^{(t),*}\})$ be an optimal solution of (2.4). 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 (2.4) are affine, then it follows from [31, 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 (4.1). Thus, the solution set of the KKT system (4.1) 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 (4.1). 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 (1.1), one can verify that $\big(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}, \boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ satisfies the KKT system (4.2). Hence, the solution set of the KKT system (4.2) is nonempty. Moreover, from [31, Theorem 3.27], we see that $\big(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\}\big)$ is also an optimal solution of (2.6). This shows that the optimal solution of (2.6) exists and proves statement (ii).

Statement (iii). First, it is easy to see from [31, Theorem 3.27] that $(\boldsymbol{u}^*, \{V^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ solves (2.6). Then, simplifying the KKT system (4.2) and recalling (1.1), one can verify that $(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}, \{\boldsymbol{y}^{(t),*}\}, \{\boldsymbol{z}^{(t),*}\})$ satisfies the KKT system (4.1) with $\boldsymbol{\lambda}^*$ in place of $\boldsymbol{w}$ and $\Lambda^{(t),*}$ in place of $\Pi^{(t)}$. Now, using [31, Theorem 3.27] again, we see that $\big(\boldsymbol{\lambda}^*, \{\Lambda^{(t),*}\}\big)$ is an optimal solution of (2.4). This proves statement (iii). $\square$

In order to present the global convergence of Algorithm 1 based on the theory developed in [25], we first express (2.6) as follows:

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

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

$$A = \begin{bmatrix} -I_m & \\ & I_{m \sum_{t=1}^N m_t} \end{bmatrix}, \; B_1 = \begin{bmatrix} 1 & \cdots & 1 \\ -\boldsymbol{e}_{m_1} & & \\ & \ddots & \vdots \\ & & -\boldsymbol{e}_{m_N} \end{bmatrix} \otimes I_m, \; B_2 = \begin{bmatrix} 0 & \cdots & 0 \\ -I_{m_1} & & \\ & \ddots & \vdots \\ & & -I_{m_N} \end{bmatrix} \otimes \boldsymbol{e}_m. \quad (4.3)$$

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

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

By using the above notation, the problem (2.6) can be rewritten in a compact form as follows:

$$\min\ \theta(\boldsymbol{u}, \boldsymbol{v}) + g(\boldsymbol{y}, \boldsymbol{z}) \qquad \text{s.t.} \qquad A[\boldsymbol{u}; \boldsymbol{v}] + B[\boldsymbol{y}; \boldsymbol{z}] = \boldsymbol{d}, \tag{4.4}$$

where $B = [B_1\ B_2]$. Our sGS-ADMM (Algorithm 1) is precisely a 2-block sPADMM applied to the compact form (4.4) of (2.6) with a specially designed proximal term. In particular, **Step 1** of the algorithm is the same as computing

$$(\boldsymbol{u}^{k+1},\, \boldsymbol{v}^{k+1}) \ =\ \mathrm{argmin}_{\boldsymbol{u},\boldsymbol{v}} \left\{ \mathcal{L}_\beta(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y}^k, \boldsymbol{z}^k; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}) \right\}. \tag{4.5}$$

It follows from [25, Proposition 5] that **Step 2a**–**2c** is equivalent to

$$(\boldsymbol{y}^{k+1},\, \boldsymbol{z}^{k+1}) \ =\ \mathrm{argmin}_{\boldsymbol{y},\boldsymbol{z}} \big\{ \mathcal{L}_\beta(\boldsymbol{u}^{k+1}, \boldsymbol{v}^{k+1}, \boldsymbol{y}, \boldsymbol{z}; \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}) + \tfrac{1}{2}\|[\boldsymbol{y}; \boldsymbol{z}] - [\boldsymbol{y}^k; \boldsymbol{z}^k]\|_C^2 \big\}, \tag{4.6}$$

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

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

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

Theorem 4.1. *Let $\tau \in (0, \frac{1+\sqrt{5}}{2})$ and $\big\{\big(\boldsymbol{u}^k, \{V^{(t),k}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}, \boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)\big\}$ be the sequence generated by the sGS-ADMM in Algorithm 1. Then, the sequence $\big\{\big(\boldsymbol{u}^k, \{V^{(t),k}\}, \{\boldsymbol{y}^{(t),k}\}, \{\boldsymbol{z}^{(t),k}\}\big)\big\}$ converges to an optimal solution of (2.6) and the sequence $\big\{\big(\boldsymbol{\lambda}^k, \{\Lambda^{(t),k}\}\big)\big\}$ converges to an optimal solution of (2.4).*

Proof. Here we apply the convergence result developed in [15] to the 2-block sPADMM outlined in (4.5), (4.6) and **Step 3** of Algorithm 1. Since both $A^\top A$ and $C + \beta B^\top B$ are positive definite, the conditions for ensuring the convergence of the 2-block sPADMM in [15, Theorem B.1] are satisfied, thus along with Proposition 4.1, one can readily apply [15, Theorem B.1] to obtain the desired results. □

Based on the equivalence of our sGS-ADMM to a 2-block sPADMM, the linear convergence rate of the sGS-ADMM can be established from the linear convergence result of the 2-block sPADMM; see [19, Section 4.1] for more details. Thus, following these results, we can derive the global linear convergence rate of our sGS-ADMM in Algorithm 1.

Define

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

where A, B_1, B_2 are defined in (4.3) and $s_\tau := (5 - \tau - 3\min\{\tau, \tau^{-1}\})/4$. 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 (4.2). Recall from Proposition 4.1(ii) that Ω is nonempty. Moreover, for any $\mathcal{W} \in \mathscr{W}$, denote

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

We then present the linear convergence result of Algorithm 1 in the following theorem.

Theorem 4.2. *Let $\tau \in (0, \frac{1+\sqrt{5}}{2})$ 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 $k \geq 1$,*

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

Proof. First we note the equivalence of the sGS-ADMM in Algorithm 1 to a 2-block sPADMM. Next consider the KKT mapping $\mathcal{R} : \mathscr{W} \to \mathscr{W}$ defined by

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

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 not hard to see that $\mathcal{R}(\cdot)$ is continuous on $\mathscr{W}$ and $\mathcal{R}(\mathcal{W}) = 0$ if and only if $\mathcal{W} \in \Omega$. By Theorem 4.1, 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 [30, 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 [28], 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

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

Finally, based on the above facts and Proposition 4.1, we can apply [19, Corollary 1] to obtain the desired results. □

5 Extension to the free support case

In this section, we briefly discuss the case that the support points of a barycenter are not pre-specified and hence one needs to solve (2.1) to find a barycenter. Note that problem (2.1) 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 (2.1). Specifically, with X being fixed, problem (2.1) indeed reduces to (2.2) (hence (2.4)), then one can call our sGS-ADMM in Algorithm 1 as a subroutine to solve it efficiently. On the other hand, with $(\boldsymbol{w}, \{\Pi^{(t)}\})$ being fixed, problem (2.1) reduces to a simple quadratic optimization problem with respect to X and one can easily obtain the optimal X^* by

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

In fact, this alternating minimization strategy has also been used in [13, 36, 37] to handle the free support case by using their proposed methods as subroutines. The complete algorithm for solving (2.1) is presented in Algorithm 2.

Algorithm 2 Alternating minimization method for solving (2.1)

Input: Choose an initial X^0 arbitrarily. Set $s = 0$.

while a termination criterion is not met, **do**

Step 1. Apply the sGS-ADMM in Algorithm 1 to find an optimal solution $(\boldsymbol{w}^{s+1}, \{\Pi^{(t),s+1}\})$ of (2.4) with $X = X^s$.

Step 2. Compute X^{s+1} by $\boldsymbol{x}_i^* = \left(\sum_{t=1}^N \sum_j^{m_t} \pi_{ij}^{(t)}\right)^{-1} \sum_{t=1}^N \sum_j^{m_t} \pi_{ij}^{(t)} \boldsymbol{q}_j^{(t)}$ for $i = 1, \cdots, N$.

Step 3. Set $s = s + 1$ and go to **Step 1**.

end while

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

6 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 (2.2). We also compare our sGS-ADMM with two existing representative methods, namely, the iterative Bregman projection (IBP) method [4] and the modified Bregman ADMM (BADMM) [37]. 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 E5-2680@2.50GHz (this processor has 12 cores and 24 threads) and 128GB of RAM, equipped with 64-bit Windows 10 OS.

In our implementation of the sGS-ADMM, a data scaling technique is used. Let $\kappa = 1/\|[D^{(1)}, \cdots, D^{(N)}]\|_F$. Then, problem (2.2) is equivalent to

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\, \{\Pi^{(t)}\}} & \sum_{t=1}^N \langle \hat{D}^{(t)},\, \Pi^{(t)} \rangle \\ \text{s.t.} & \Pi^{(t)} \boldsymbol{e}_{m_t} = \boldsymbol{w},\ (\Pi^{(t)})^\top \boldsymbol{e}_m = \boldsymbol{a}^{(t)},\ \Pi^{(t)} \geq 0, \quad \forall\, t = 1, \cdots, N, \\ & \boldsymbol{e}_m^\top \boldsymbol{w} = 1,\ \boldsymbol{w} \geq 0, \end{array} \tag{6.1}$$

where $\hat{D}^{(t)} = \kappa D^{(t)}$ for $t = 1, \cdots, N$. We then apply the sGS-ADMM to solve the dual problem of (6.1) to obtain an optimal solution of (2.2). Indeed, this technique has been widely used in the ADMM-based methods to improve their numerical performance. Its effectiveness has also been observed in our experiments.

For a set of vectors $\{\boldsymbol{a}^{(t)} \mid t = 1, \ldots, N\}$, we define the notation $\|\{\boldsymbol{a}^{(t)}\}\| = \left(\sum_{t=1}^N \|\boldsymbol{a}^{(t)}\|^2\right)^{\frac{1}{2}}$. Similarly, for a set of matrices $\{A^{(t)} \mid t = 1, \ldots, N\}$, we define the notation $\|\{A^{(t)}\}\|_F = \left(\sum_{t=1}^N \|A^{(t)}\|_F^2\right)^{\frac{1}{2}}$. 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 (4.2) as follows:

$$\eta_1(\boldsymbol{\lambda}, \boldsymbol{u}) = \tfrac{\|\boldsymbol{\lambda} - \mathrm{Pr}_{\Delta_m}(\boldsymbol{\lambda}+\boldsymbol{u})\|}{1+\|\boldsymbol{\lambda}\|+\|\boldsymbol{u}\|}, \qquad \eta_2(\{V^{(t)}\}, \{\Lambda^{(t)}\}) = \tfrac{\|\{V^{(t)} - (V^{(t)} - \Lambda^{(t)})_+\}\|_F}{1+\|\{V^{(t)}\}\|_F+\|\{\Lambda^{(t)}\}\|_F},$$

$$\eta_3(\boldsymbol{\lambda}, \{\Lambda^{(t)}\}) = \tfrac{\|\{\Lambda^{(t)} \boldsymbol{e}_{m_t} - \boldsymbol{\lambda}\}\|}{1+\|\boldsymbol{\lambda}\|+\|\{\Lambda^{(t)}\}\|_F}, \qquad \eta_4(\{\Lambda^{(t)}\}) = \tfrac{\|\{(\Lambda^{(t)})^\top \boldsymbol{e}_m - \boldsymbol{a}^{(t)}\}\|_F}{1+\|\{\boldsymbol{a}^{(t)}\}\|+\|\{\Lambda^{(t)}\}\|_F},$$

$$\eta_5(\boldsymbol{u}, \{\boldsymbol{y}^{(t)}\}) = \tfrac{\|\sum_{t=1}^N \boldsymbol{y}^{(t)} - \boldsymbol{u}\|}{1+\|\sum_{t=1}^N \boldsymbol{y}^{(t)}\|+\|\boldsymbol{u}\|}, \quad \eta_6(\{V^{(t)}\}, \{\boldsymbol{y}^{(t)}\}, \{\boldsymbol{z}^{(t)}\}) = \tfrac{\|\{V^{(t)} - D^{(t)} - \boldsymbol{y}^{(t)} \boldsymbol{e}_{m_t}^\top - \boldsymbol{e}_m (\boldsymbol{z}^{(t)})^\top\}\|_F}{1+\|\{D^{(t)}\}\|_F+\|\{V^{(t)}\}\|_F+\|\{\boldsymbol{y}^{(t)}\}\|+\|\{\boldsymbol{z}^{(t)}\}\|},$$

$$\eta_7(\boldsymbol{\lambda}) = \tfrac{|\boldsymbol{e}_m^\top \boldsymbol{\lambda} - 1| + \|\min(\boldsymbol{\lambda}, 0)\|}{1+\|\boldsymbol{\lambda}\|}, \qquad \eta_8(\{\Lambda^{(t)}\}) = \tfrac{\|\min([\Lambda^{(1)}, \cdots, \Lambda^{(N)}], 0)\|_F}{1+\|\{\Lambda^{(t)}\}\|_F}.$$

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

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

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 (4.2). 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

$$\max\big\{\eta_P(\mathcal{W}^{k+1}),\, \eta_D(\mathcal{W}^{k+1}),\, \eta_{gap}(\mathcal{W}^{k+1})\big\} < 10^{-5}, \tag{6.2}$$

where $\mathcal{W}^{k+1}$ is generated by the sGS-ADMM at the k-th iteration. Furthermore, the maximum number of iterations for the sGS-ADMM is set to be 3000.

We also use a similar numerical strategy as [21, Section 4.4] to update the penalty parameter β in the augmented Lagrangian function at every 50 iterations to improve the convergence speed of the sGS-ADMM. Initially, set $\beta_0 = 1$. At the k-th iteration, compute $\chi^{k+1} = \frac{\eta_D(\mathcal{W}^{k+1})}{\eta_P(\mathcal{W}^{k+1})}$ and then, set

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

Note that computing all 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, we follow [4, Remark 3] to implement and terminate the algorithm when

$$\frac{\|\boldsymbol{w}^{k+1}-\boldsymbol{w}^k\|}{1+\|\boldsymbol{w}^{k+1}\|+\|\boldsymbol{w}^k\|} < 10^{-8}, \quad \frac{\|\{\boldsymbol{u}^{(t),k+1}-\boldsymbol{u}^{(t),k}\}\|}{1+\|\{\boldsymbol{u}^{(t),k}\}\|+\|\{\boldsymbol{u}^{(t),k+1}\}\|} < 10^{-8}, \quad \frac{\|\{\boldsymbol{v}^{(t),k+1}-\boldsymbol{v}^{(t),k}\}\|_F}{1+\|\{\boldsymbol{v}^{(t),k}\}\|+\|\{\boldsymbol{v}^{(t),k+1}\}\|} < 10^{-8},$$

where $(\boldsymbol{w}^{k+1}, \{\boldsymbol{u}^{(t),k+1}\}, \{\boldsymbol{v}^{(t),k+1}\})$ is generated at the k-th iteration (see Appendix A). The maximum number of iterations for IBP is set to be 10000. Moreover, as in [4], the regularization parameter ε is chosen from $\{0.1, 0.01, 0.001\}$ in our experiments.

For BADMM, we use the Matlab codes[1] implemented by the authors in [37] and terminate it when

$$\max\{\eta_3(\boldsymbol{w}^{k+1}, \{\Gamma^{(t),k+1}\}),\, \eta_4(\{\Pi^{(t),k+1}\})\} < 10^{-5}, \quad \frac{\|\boldsymbol{w}^{k+1}-\boldsymbol{w}^k\|}{1+\|\boldsymbol{w}^{k+1}\|+\|\boldsymbol{w}^k\|} < 10^{-5},$$

$$\frac{\|\{\Pi^{(t),k+1}-\Gamma^{(t),k+1}\}\|_F}{1+\|\{\Pi^{(t),k+1}\}\|_F+\|\{\Gamma^{(t),k+1}\}\|_F} < 10^{-5}, \quad \frac{\|\{\Pi^{(t),k+1}-\Pi^{(t),k}\}\|_F}{1+\|\{\Pi^{(t),k}\}\|_F+\|\{\Pi^{(t),k+1}\}\|_F} < 10^{-5},$$

$$\frac{\|\{\Gamma^{(t),k+1}-\Gamma^{(t),k}\}\|_F}{1+\|\{\Gamma^{(t),k}\}\|_F+\|\{\Gamma^{(t),k+1}\}\|_F} < 10^{-5}, \quad \frac{\|\{\Lambda^{(t),k+1}-\Lambda^{(t),k}\}\|_F}{1+\|\{\Lambda^{(t),k}\}\|_F+\|\{\Lambda^{(t),k+1}\}\|_F} < 10^{-5},$$

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). The above termination criteria is checked at every 200 iterations and the maximum number of iterations for BADMM is set to be 3000.

6.1 Experiments on synthetic data

In this subsection, we generate a set of discrete probability distributions $\{\mathcal{P}^{(t)}\}_{t=1}^N$ with $\mathcal{P}^{(t)} = \{(a_i^{(t)}, \boldsymbol{q}_i^{(t)}) \in \mathbb{R}_+ \times \mathbb{R}^d : i = 1, \cdots, m_t\}$ and $\sum_{i=1}^{m_t} a_i^{(t)} = 1$, and then apply different methods to solve (2.2) to compute a Wasserstein barycenter $\mathcal{P} = \{(w_i, \boldsymbol{x}_i) \in \mathbb{R}_+ \times \mathbb{R}^d : i = 1, \cdots, m\}$, where m and $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m)$ are pre-specified.

In the following experiments, we set $d = 3$ and $m_1 = \cdots = m_N = m'$ for convenience, and then choose different (N, m, m') with $m \ge m'$. Given each triple (N, m, m'), we randomly generate a trial in the following three cases.

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

- **Case 1.** Each distribution has different *dense* support points. In this case, for each $\mathcal{P}^{(t)}$, we first generate the support points $(\boldsymbol{q}_1^{(t)}, \cdots, \boldsymbol{q}_{m'}^{(t)})$ with i.i.d. standard Gaussian entries. We next 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[2] 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* support points. In this case, for each $\mathcal{P}^{(t)}$, we also generate the support points $(\boldsymbol{q}_1^{(t)}, \cdots, \boldsymbol{q}_{m'}^{(t)})$ with i.i.d. standard Gaussian entries. We next 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 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 $\lfloor m' \times \mathtt{sr} \rfloor$, where $\mathtt{sr}$ 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 $\{\boldsymbol{q}_i^{(t)} : a_i^{(t)} \neq 0, t = 1, \cdots, N\}$ by the k-means method. Note that, in this case, one can actually solve a smaller problem (2.3) to obtain an optimal solution of (2.2); see Remark 2.1.

- **Case 3.** All distributions have the *same* support points. In this case, we set $m = m'$ and generate the points $(\boldsymbol{q}_1, \cdots, \boldsymbol{q}_m)$ with i.i.d. standard Gaussian entries. 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 $\mathcal{P}^{(t)}$, 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 $\sum_{i=1}^{m} a_i^{(t)} = 1$.

For each trial, we also apply Gurobi 8.0.0 [18] (with an academic license) to solve (2.2). It is well known that Gurobi is a very powerful commercial package for solving linear programming problems and can provide high quality solutions. Therefore, we use Gurobi as a benchmark to evaluate the performance of different methods. Moreover, for Gurobi, we use the default (also the optimal) parameter settings (see [18] for more details) so that Gurobi can exploit multiple processors in our workstation, while other methods including our sGS-ADMM are implemented without parallelism or concurrency. Note also that we used Gurobi to solve the problems to the default accuracy level of 1e-8. We observed that the time taken by Gurobi to solve the problems to the accuracy level of 1e-6 is only marginally shorter because Gurobi employs a cross-over strategy (when the iterates are deemed close enough to an optimal solution) that allows it to solve the problems to a higher accuracy very quickly.

Tables 1, 2, 3 present the numerical results of different methods for **Cases 1**, **2**, **3**, respectively, where we use different choices of (N, m, m') and different sparsity ratio $\mathtt{sr}$. In these tables, "normalized obj" denotes the normalized objective value defined by $\big|\mathcal{F}(\{\Pi^{(t),*}\}) - \mathcal{F}_{\rm gu}\big| / \mathcal{F}_{\rm gu}$, 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 $\mathcal{F}_{\rm gu}$ denotes the objective value obtained by Gurobi; "feasibility" denotes the value of

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

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 (since Gurobi use a hybrid method, then we do not report its number of iterations here and use "–" instead). All the 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 IBP and BADMM in the sense that it always returns an objective value that is considerably closer to that of Gurobi while achieving comparable feasibility accuracy in less computational time. For IBP, we see that it gives a better feasibility accuracy than sGS-ADMM and BADMM in most cases, especially when ε is large. Moreover, when all distributions have the *same* support points, IBP takes much less computational time than the other algorithms; see Table 3. This is because IBP can be implemented very efficiently

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

in this case thanks to the favorable iterative scheme (see Appendix A). However, the objective value returned by IBP is always far away from that of Gurobi, which means that the quality of the solution obtained by IBP is not good. This is because IBP only solves an approximate problem of (2.2) and the regularization parameter ε cannot be too small (hence IBP with $\varepsilon = 0.001$ performs worst). For BADMM, one can see that it is able to give an objective value close to that of Gurobi. However, it takes much more time and its feasibility accuracy is the worst for all 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 indeed solves 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 variables with nonnegative constraints and 50001 equality constraints. In this case, we see that Gurobi is about 10 times slower than our sGS-ADMM.

Table 1: Numerical results on synthetic data for the case that each distribution has different *dense* support points. In the table, "a" stands for Gurobi; "b" stands for sGS-ADMM; "c" stands for BADMM; "d1" stands for IBP with $\varepsilon = 0.1$; "d2" stands for IBP with $\varepsilon = 0.01$; "d3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	a	b	c	d1	d2	d3	a	b	c	d1	d2	d3
			normalized obj						feasibility					
20	100	100	0	1.11e-4	1.62e-4	1.29e+0	2.08e-1	5.78e+0	1.13e-8	1.38e-5	1.48e-4	7.34e-9	2.02e-8	2.79e-5
20	200	100	0	1.73e-4	2.22e-4	1.54e+0	3.15e-1	6.02e+0	6.84e-8	1.40e-5	1.74e-4	1.20e-8	3.20e-8	8.12e-6
20	200	200	0	1.25e-4	8.28e-5	2.16e+0	2.30e-1	4.64e+0	1.70e-7	1.40e-5	1.76e-4	7.54e-9	1.04e-7	1.92e-5
20	300	200	0	2.18e-4	1.31e-4	2.31e+0	2.76e-1	4.52e+0	7.17e-8	1.40e-5	2.00e-4	1.52e-9	2.68e-7	1.12e-5
50	100	100	0	6.67e-5	9.51e-5	1.14e+0	1.46e-1	3.59e+0	6.32e-8	1.39e-5	2.29e-4	1.28e-8	1.22e-7	9.16e-5
50	200	100	0	1.31e-4	1.40e-4	1.47e+0	2.77e-1	7.47e+0	2.20e-7	1.40e-5	2.41e-4	5.11e-9	5.63e-8	2.24e-5
50	200	200	0	1.61e-4	1.50e-4	2.05e+0	1.93e-1	1.85e+0	2.94e-7	1.41e-5	2.63e-4	3.03e-8	1.33e-7	3.48e-5
50	300	200	0	2.66e-4	2.46e-4	2.26e+0	2.60e-1	2.41e+0	3.98e-7	1.40e-5	3.01e-4	1.34e-9	1.00e-7	1.41e-5
100	100	100	0	1.25e-4	5.37e-5	1.09e+0	1.22e-1	1.12e+0	4.46e-8	1.42e-5	3.01e-4	1.14e-8	2.32e-7	1.45e-4
100	200	100	0	2.34e-4	9.46e-5	1.42e+0	2.57e-1	4.01e+0	1.53e-7	1.40e-5	3.29e-4	2.63e-9	6.45e-8	3.55e-5
100	200	200	0	2.79e-4	2.41e-4	1.98e+0	1.71e-1	8.54e-1	3.10e-7	1.49e-5	3.62e-4	9.40e-9	9.71e-8	5.42e-5
100	300	200	0	3.90e-4	3.02e-4	2.24e+0	2.52e-1	7.38e+0	1.60e-7	1.44e-5	3.88e-4	5.45e-9	2.24e-7	2.16e-5
			iter						time (in seconds)					
20	100	100	–	2545	3000	143	3137	10000	2.33	3.92	48.46	0.36	6.87	30.23
20	200	100	–	2465	3000	108	2237	10000	7.39	7.14	98.41	0.49	8.96	58.31
20	200	200	–	2575	3000	139	3011	10000	12.43	16.20	197.66	1.13	22.85	114.31
20	300	200	–	2515	3000	102	1937	10000	25.68	23.57	296.46	1.25	22.28	171.87
50	100	100	–	2850	3000	127	4590	10000	9.32	10.40	120.54	0.69	22.71	70.14
50	200	100	–	2745	3000	113	2687	10000	53.86	21.82	247.78	1.17	25.99	136.29
50	200	200	–	2885	3000	114	3916	10000	65.58	46.61	494.19	2.34	77.04	289.80
50	300	200	–	2805	3000	132	2748	10000	168.43	73.95	748.56	4.31	86.43	464.63
100	100	100	–	2980	3000	174	7693	10000	13.66	23.97	248.07	1.80	76.62	138.14
100	200	100	–	2860	3000	111	5345	10000	38.57	47.10	495.83	2.31	107.24	283.78
100	200	200	–	3000	3000	103	4479	10000	70.83	105.21	990.02	4.54	189.32	615.58
100	300	200	–	3000	3000	120	3867	9926	1575.34	157.88	1546.77	8.33	254.73	1012.52

To further compare the performance of Gurobi and our sGS-ADMM, we conduct more experiments on synthetic data for **Case 1**, where we set $m = 20$, $m' = 10$ and the number of samples to large values, say $N \in \{5000, 10000, 20000, 40000, 60000, 80000\}$. In this part of experiments, we only use (6.2) to terminate our sGS-ADMM without setting the maximum iteration number. Figure 1 shows the running time across a wide range of N by the two algorithms 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 to Gurobi and has a comparable feasibility accuracy. For the computational time, our sGS-ADMM increases linearly with respect to the number of samples, while Gurobi increases much faster. This is because the solution methods used in Gurobi (the primal/dual simplex method and the barrier method) are no longer efficient 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 concurrent optimization strategy to exploit multiple processors. 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 m and m' are fixed, the

Table 2: Numerical results on synthetic data for the case that each distribution has different *sparse* support points. In the table, "a" stands for Gurobi; "b" stands for sGS-ADMM; "c" stands for BADMM; "d1" stands for IBP with $\varepsilon = 0.1$; "d2" stands for IBP with $\varepsilon = 0.01$; "d3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	sr	a	b	c	d1	d2	d3	a	b	c	d1	d2	d3
				normalized obj						feasibility					
50	50	500	0.1	0	3.90e-5	2.18e-4	6.24e-1	1.25e-1	2.48e+0	4.83e-8	1.38e-5	2.12e-4	7.76e-9	2.62e-7	2.30e-4
50	100	500	0.2	0	7.65e-5	1.10e-4	1.16e+0	1.50e-1	3.46e+0	1.41e-15	1.40e-5	2.48e-4	8.41e-9	5.35e-8	9.98e-5
50	100	1000	0.1	0	6.24e-5	8.86e-5	1.14e+0	1.39e-1	9.64e-1	9.02e-8	1.39e-5	2.29e-4	1.76e-9	1.31e-7	1.11e-4
50	200	1000	0.2	0	1.74e-4	1.80e-4	2.03e+0	1.85e-1	3.24e+0	2.45e-7	1.41e-5	2.74e-4	1.03e-8	1.09e-7	2.82e-5
100	50	500	0.1	0	4.86e-5	1.80e-4	5.48e-1	8.55e-2	2.15e+0	3.61e-8	1.41e-5	2.85e-4	2.74e-8	4.80e-6	2.73e-4
100	100	500	0.2	0	1.36e-4	7.51e-5	1.10e+0	1.26e-1	5.59e+0	1.15e-7	1.42e-5	2.84e-4	7.85e-9	5.41e-7	1.35e-4
100	100	1000	0.1	0	1.27e-4	7.58e-5	1.09e+0	1.21e-1	1.12e+0	4.55e-8	1.43e-5	3.02e-4	1.76e-8	5.54e-7	1.54e-4
100	200	1000	0.2	0	3.16e-4	2.19e-4	2.01e+0	1.79e-1	8.69e-1	5.12e-7	1.49e-5	3.54e-4	9.43e-9	1.23e-7	5.33e-5
200	50	500	0.1	0	8.63e-5	1.39e-4	5.22e-1	7.38e-2	5.41e-1	1.08e-7	1.41e-5	2.93e-4	6.62e-8	1.43e-6	4.09e-4
200	100	500	0.2	0	2.61e-4	4.98e-5	1.05e+0	1.10e-1	4.48e-1	1.59e-7	1.52e-5	3.77e-4	2.62e-8	9.73e-7	1.92e-4
200	100	1000	0.1	0	2.63e-4	4.39e-5	1.05e+0	1.05e-1	4.16e-1	1.11e-7	1.53e-5	3.81e-4	3.22e-8	1.57e-6	1.98e-4
200	200	1000	0.2	0	7.37e-4	2.76e-4	1.93e+0	1.52e-1	3.27e-1	4.11e-7	1.61e-5	4.52e-4	1.03e-8	7.20e-7	8.56e-5
				iter						time (in seconds)					
50	50	500	0.1	–	2700	3000	149	6819	10000	3.77	2.38	22.99	0.13	5.38	13.24
50	100	500	0.2	–	2870	3000	189	4740	10000	10.60	11.54	126.91	1.03	24.67	72.21
50	100	1000	0.1	–	2875	3000	116	5277	10000	9.50	11.35	123.28	0.65	26.87	72.01
50	200	1000	0.2	–	2890	3000	167	2840	10000	72.45	47.99	501.87	3.49	57.65	293.15
100	50	500	0.1	–	2880	3000	145	8717	10000	2.70	5.56	60.14	0.44	23.89	35.56
100	100	500	0.2	–	2985	3000	155	7351	10000	15.21	24.39	247.05	1.62	72.68	136.23
100	100	1000	0.1	–	2985	3000	104	6036	10000	13.55	24.48	246.69	1.11	59.71	138.94
100	200	1000	0.2	–	3000	3000	118	4938	10000	73.16	107.15	1005.36	5.23	211.66	624.63
200	50	500	0.1	–	3000	3000	145	9930	10000	5.66	12.19	124.58	0.83	51.89	69.11
200	100	500	0.2	–	3000	3000	149	8759	10000	39.37	50.49	495.39	3.17	180.19	285.67
200	100	1000	0.1	–	3000	3000	107	8872	10000	38.88	50.37	495.38	2.30	181.82	284.77
200	200	1000	0.2	–	3000	3000	108	6553	10000	191.10	208.23	2051.56	10.25	580.08	1435.06

total computational cost of our sGS-ADMM shall be approximately linear with respect to N, clearly evident in Figure 1. In addition, this implies the potential advantages of the parallel implementation of our sGS-ADMM. We will leave this research topic in the future.

6.2 Experiments on MNIST

In this subsection, to better visualize the performance of each method, we conduct similar experiments to [13, Section 6.1] on the MNIST[3] dataset [22]. Specifically, we randomly select 50 images for each digit $(0-9)$ and resize each image to ζ times of its original size 28×28, where ζ is drawn uniformly at random between 0.5 and 2. Then, we consider the following two cases.

- **Case 1.** We normalize each resized image so that all pixel values add up to 1. Thus, each image can be viewed as a discrete distribution. We show 10 of 50 resulting images for digit 6 in the first row of Figure 2. In this case, the input images have *different* sizes between 14×14 and 56×56, i.e., they have *different* support points.

- **Case 2.** We first randomly put each resized image in a larger 56×56 blank image and then normalize the resulting image so that all pixel values add up to 1. We show 10 of 50 resulting images for digit 6 in the second row of Figure 2. In this case, the input images have the *same* size 56×56, i.e., they have the *same* support points.

Next, for each case, we apply sGS-ADMM, BADMM and IBP ($\varepsilon = 0.01$) to compute a Wasserstein barycenter of the resulting images for each digit. The size of barycenter is set to 56×56. Moreover, since each input image can be viewed as a *sparse* discrete distribution because most of the pixel values are zeros, then one can actually solve a smaller problem (2.3) to obtain a barycenter; see Remark 2.1.

[3] Available in http://yann.lecun.com/exdb/mnist/.

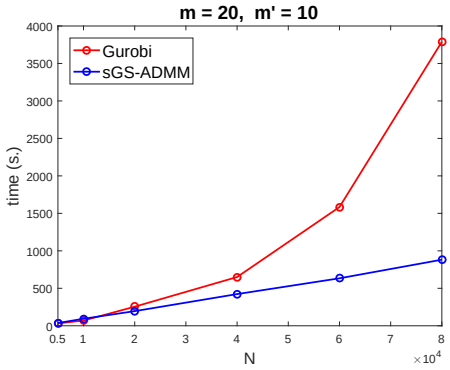
Table 3: Numerical results on synthetic data for the case that all distributions have the *same* support points. In the table, "a" stands for Gurobi; "b" stands for sGS-ADMM; "c" stands for BADMM; "d1" stands for IBP with $\varepsilon = 0.1$; "d2" stands for IBP with $\varepsilon = 0.01$; "d3" stands for IBP with $\varepsilon = 0.001$.

N	m	m'	a	b	c	d1	d2	d3	a	b	c	d1	d2	d3
			normalized obj						feasibility					
20	50	50	0	9.93e-5	6.89e-4	8.05e-1	1.05e-2	7.53e-1	3.38e-8	1.43e-5	1.58e-4	1.92e-8	2.11e-6	5.02e-4
20	100	100	0	1.95e-4	2.72e-4	1.60e+0	2.70e-2	7.18e-1	4.41e-8	1.40e-5	2.44e-4	1.73e-8	6.17e-6	3.03e-4
20	200	200	0	3.81e-4	1.47e-3	2.93e+0	6.19e-2	6.37e-1	2.26e-7	1.41e-5	3.16e-4	1.89e-8	1.39e-6	1.81e-4
20	300	300	0	8.04e-4	3.60e-3	4.20e+0	1.06e-1	5.00e-1	4.43e-7	1.41e-5	3.86e-4	6.85e-9	3.19e-7	1.19e-4
50	50	50	0	5.79e-5	6.30e-4	7.66e-1	9.76e-3	7.51e-1	1.06e-15	1.60e-5	2.41e-4	3.61e-8	1.70e-5	5.47e-4
50	100	100	0	9.79e-5	2.97e-4	1.56e+0	2.59e-2	7.23e-1	7.74e-8	1.61e-5	3.01e-4	2.76e-8	7.83e-6	3.27e-4
50	200	200	0	2.17e-4	1.56e-3	2.84e+0	5.95e-2	6.52e-1	2.76e-7	1.57e-5	4.38e-4	1.56e-8	3.38e-6	2.28e-4
50	300	300	0	2.31e-4	3.41e-3	4.08e+0	1.02e-1	4.76e-1	5.34e-7	1.64e-5	5.59e-4	9.15e-9	2.33e-6	1.41e-4
100	50	50	0	7.33e-5	7.88e-4	7.59e-1	9.51e-3	7.60e-1	4.46e-8	1.64e-5	2.65e-4	5.69e-8	1.10e-5	6.31e-4
100	100	100	0	1.82e-4	2.02e-4	1.54e+0	2.55e-2	7.21e-1	2.16e-7	1.63e-5	3.65e-4	3.33e-8	8.32e-6	4.23e-4
100	200	200	0	2.83e-4	1.64e-3	2.83e+0	5.89e-2	6.36e-1	5.19e-7	1.73e-5	5.58e-4	1.78e-8	4.03e-6	2.59e-4
100	300	300	0	5.42e-4	3.58e-3	4.01e+0	9.97e-2	4.80e-1	5.07e-7	1.78e-5	7.21e-4	1.42e-8	3.52e-6	1.97e-4
			iter						time (in seconds)					
20	50	50	–	2870	3000	530	8563	10000	0.47	1.20	10.47	0.09	1.29	1.52
20	100	100	–	2880	3000	629	9601	10000	2.40	4.47	49.09	0.13	1.79	1.89
20	200	200	–	2910	3000	372	8050	10000	13.67	18.97	204.17	0.14	2.48	3.18
20	300	300	–	2920	3000	621	6904	10000	37.63	43.90	457.30	0.31	2.96	4.44
50	50	50	–	2940	3000	681	10000	10000	1.77	2.46	21.86	0.14	1.91	1.97
50	100	100	–	3000	3000	717	10000	10000	9.73	11.30	122.45	0.24	2.94	3.03
50	200	200	–	3000	3000	728	9203	10000	72.01	49.66	499.69	0.44	4.97	5.57
50	300	300	–	3000	3000	441	8772	10000	175.33	121.28	1145.76	0.41	6.29	7.65
100	50	50	–	3000	3000	733	10000	10000	3.07	5.79	60.85	0.22	2.79	2.94
100	100	100	–	3000	3000	884	10000	10000	14.87	24.59	249.34	0.44	4.59	4.92
100	200	200	–	3000	3000	995	10000	10000	73.11	106.17	1001.90	0.91	8.40	8.73
100	300	300	–	3000	3000	410	9634	10000	1314.67	230.78	2343.32	0.66	11.56	12.99

The results for **Cases 1** and **2** are shown in Figure 3 and 4, respectively. One can see that, our sGS-ADMM can provide a clearer barycenter by using less computational time. For example, in Figure 3, the results obtained by running sGS-ADMM for 100s are already much better than those obtained by running BADMM for 800s. Similarly, the results obtained by running sGS-ADMM for 100s are better than those obtained by IBP ($\varepsilon = 0.01$) where the images look blurry. By running all the algorithms for 800s, we see that sGS-ADMM is able to produce sharper images, but the quality of the images obtained by IBP does not seem to improve. While the results obtained by BADMM have improved, the quality of the images produced is still much worse than those obtained by sGS-ADMM. For Figure 4, again the performance of sGS-ADMM is much better than BADMM in terms of the image quality obtained. For IBP, the quality of the images obtained is much better than those obtained in Figure 3, but they are not as sharp as those obtained by sGS-ADMM.

7 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 show 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 dataset. Finally, we have conducted detailed numerical experiments on synthetic datasets and image datasets to illustrate the efficiency of our method. Based on the numerical results, we can see that our sGS-ADMM outperforms the powerful commercial solver Gurobi in solving large-scale LPs arising



	normalized obj		feasibility	
N	a	b	a	b
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

Figure 1: Numerical results on synthetic data for the case that each distribution has different *dense* support points. In the table, "a" stands for Gurobi; "b" stands for sGS-ADMM.

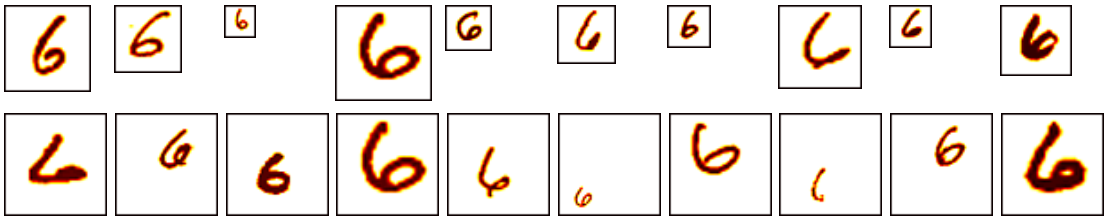


Figure 2: 10 of 50 input images for digit 6 are shown. In the first row, the input images have different sizes between 14×14 and 56×56, i.e., they have different support points. In the second row, another set of input images have the same size 56×56, i.e., they have the same support points.

from Wasserstein barycenter problems. Also, sGS-ADMM is much more efficient than the algorithm BADMM that is designed to solve the primal LP (2.4). In comparing the speed of sGS-ADMM with IBP($\varepsilon = 0.01$), our sGS-ADMM is faster when the distributions have different support points, while the latter is faster when the distributions have common support points. However, the numerical results obtained for the MNIST dataset also demonstrated that the results obtained by IBP may not be as high quality as those obtained by sGS-ADMM because the former algorithm does not solve the original LP (2.4) but an approximate entropy regularized version.

Appendix A An iterative Bregman projection

The iterative Bregman projection (IBP) was adapted in [4] to solve the following problem, which introduces an entropic regularization in the original LP (2.2):

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

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 [4, Remark 3] that IBP for solving (A.1) is given by

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

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 $\mathcal{O}(m\sum_{t=1}^N m_t)$.

It is worth noting that 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. However, we should be mindful that IBP only solves (A.1) to obtain an approximate solution of (2.2). Although a smaller ε can give a better approximation, IBP may become numerical unstable when ε is too small; see [4, Section 1.3] for more details. This situation is also observed in our experiments; see, Tables 1, 2, 3.

Appendix B A modified Bregman ADMM

The Bregman ADMM (BADMM) was first proposed in [35] and then was adapted to solve (2.2) in [37]. 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 (2.2) can be equivalently rewritten as

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

The iterative scheme of BADMM for solving (B.1) is given by

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

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 [37, Section III.B]

for more details. Indeed, at the k-th iteration,

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

Moreover, in order to avoid computing the geometric mean $(\prod_{t=1}^N \tilde{\boldsymbol{w}}^{(t),k+1})^{\frac{1}{N}}$ for updating $\boldsymbol{w}^{k+1}$, the authors in [37] actually use one of the following heuristic rules to update $\boldsymbol{w}^{k+1}$:

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

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

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

[2] 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.

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

[4] 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.

[5] J. Bigot and T. Klein. Characterization of barycenters in the Wasserstein space by averaging optimal transport maps. *To appear in ESAIM: Probability and Statistics*, 2017.

[6] S. Borgwardt and S. Patterson. Improved linear programs for discrete barycenters. *arXiv preprint arXiv: 1803.11313*, 2018.

[7] 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.

[8] 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.

[9] 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. *arXiv preprint arXiv:1803.10803*, 2018.

[10] 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.

[11] S. Claici, E. Chien, and J. Solomon. Stochastic Wasserstein barycenters. *arXiv preprint arXiv: 1802.05757*, 2018.

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

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

[14] J. A. De Loera and E. D. Kim. Combinatorics and geometry of transportation polytopes: An update. *arXiv preprint arXiv: 1307.0124*, 2013.

[15] 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.

[16] 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.

[17] 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 linéares. *Revue Francaise d'Automatique, Informatique, Recherche Opérationelle*, 9(R-2):41–76, 1975.

[18] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2018.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] X. Li, D. F. Sun, and K.-C. Toh. QSDPNAL: A two-phase augmented lagrangian method for convex quadratic semidefinite programming. *To appear in Mathematical Programming Computation*, 2018.

[27] 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.

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

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

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

[31] A. Ruszczyński. *Nonlinear Optimization*. Princeton University Press, Princeton, 2006.

[32] 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.

[33] C. A. Uribe, D. Dvinskikh, P. Dvurechensky, A. Gasnikov, and A. Nedić. Distributed computation of Wasserstein barycenters over networks. *arXiv preprint arXiv: 1803.02933*, 2018.

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

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

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

[37] 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.

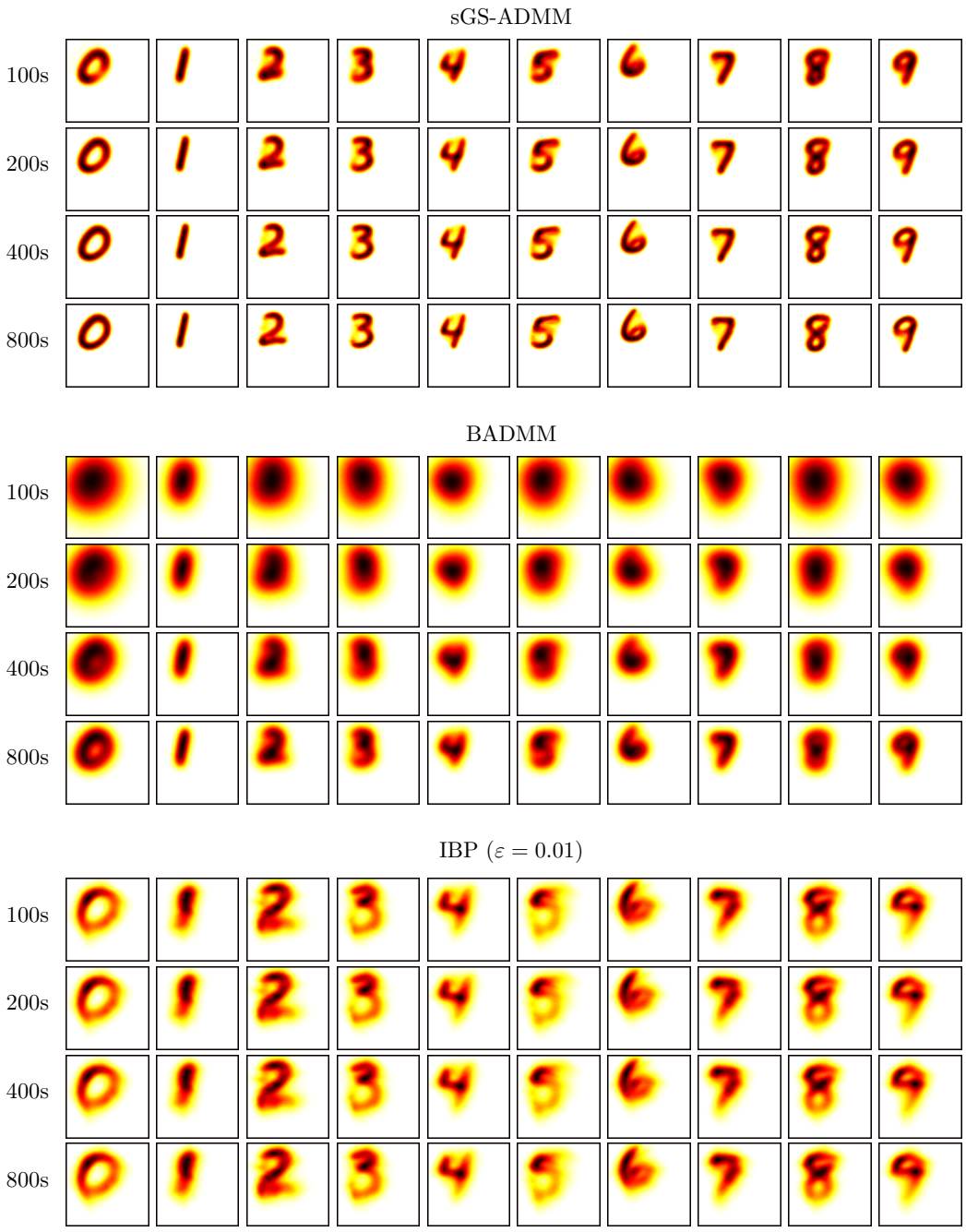


Figure 3: The barycenters obtained by running different methods for 100s, 200s, 400s, 800s, respectively. The input images have *different* sizes, i.e., they have *different* support points.

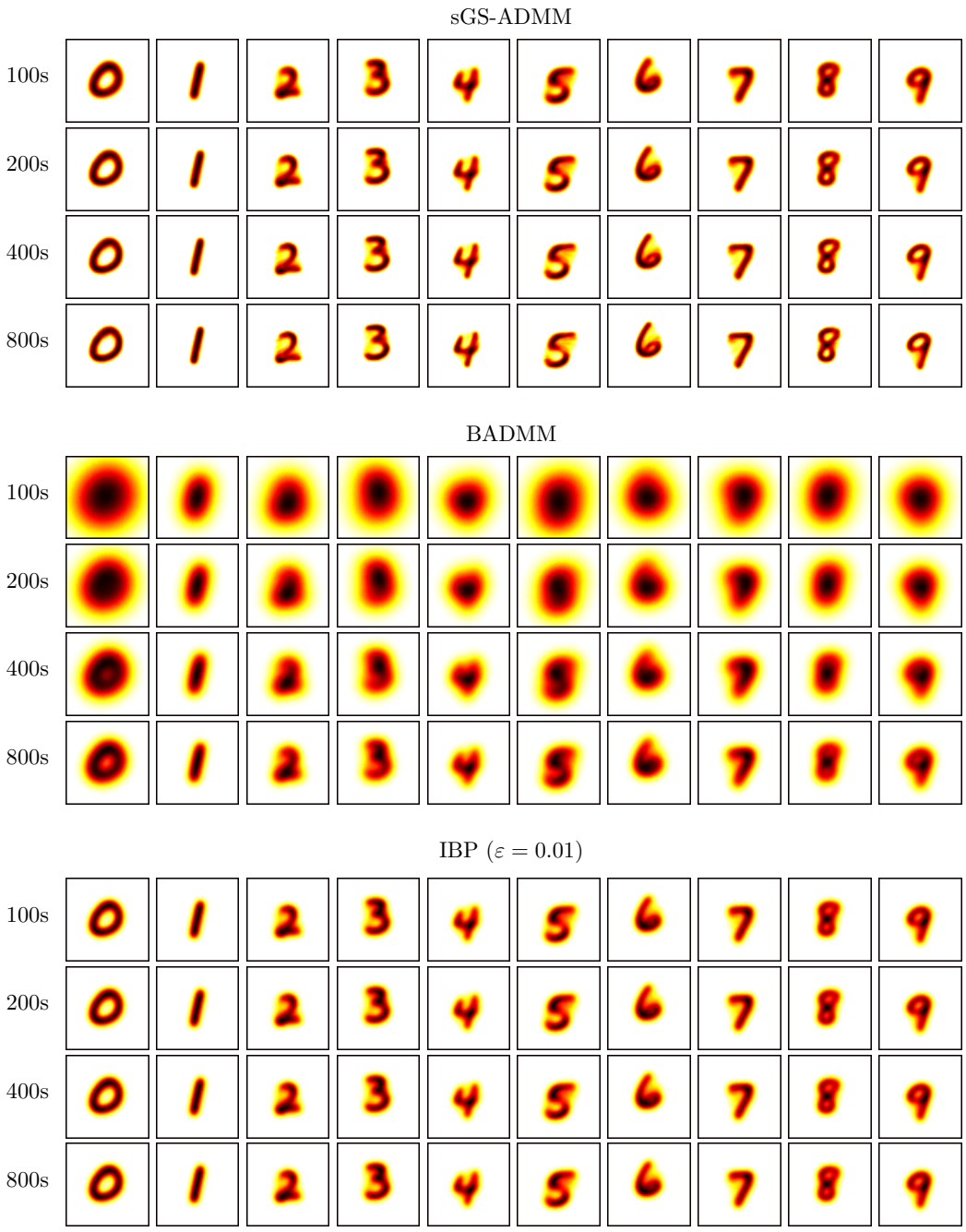


Figure 4: The barycenters obtained by running different methods for 100s, 200s, 400s, 800s, respectively. The input images have the *same* size, i.e., they have the *same* support points.